目　录

第一篇　神经网络控制及其MATLAB实现

第一篇　神经网络控制及其 MATLAB 实现

第 1 章　神经网络控制理论

人脑是一部不寻常的智能机，它能以惊人的高速度解释感觉器官传来的含糊不清的信息。它能觉察到喧闹房间内的窃窃私语，能够识别出光线暗淡的胡同中的一张面孔，更能通过不断地学习而产生伟大的创造力。古今中外，许许多多科学家为了揭开大脑机能的奥秘，从不同的角度进行着长期的不懈努力和探索，逐渐形成了一个多学科交叉的前沿技术领域——神经网络（Neural Network）。

人工神经系统的研究可以追溯到 1800 年 Frued 的精神分析学时期，他已经做了一些初步工作。1913 年，人工神经系统的第一个实践是由 Russell 描述的水力装置。1943 年，美国心理学家 Warren S McCulloch 与数学家 Walter H Pitts 合作，用逻辑的数学工具研究客观事件在形式神经网络中的描述，从此开创了对神经网络的理论研究。他们在分析、总结神经元基本特性的基础上，首先提出神经元的数学模型，简称 MP 模型。从脑科学研究来看，MP 模型不愧为第一个用数理语言描述脑的信息处理过程的模型。后来，MP 模型经过数学家的精心整理和抽象，最终发展成一种有限自动机理论，再一次展现了 MP 模型的价值，此模型沿用至今，直接影响着这一领域研究的进展。1949 年，心理学家 D.O.Hebb 提出关于神经网络学习机理的“突触修正假设”，即突触联系效率可变的假设，现在多数学习机仍遵循 Hebb 学习规则。1957 年，Frank Rosenblatt 首次提出并设计制作了著名的感知机（Perceptron），第一次从理论研究转入过程实现阶段，掀起了研究人工神经网络的高潮。今天，随着科学技术的迅猛发展，神经网络正以极大的魅力吸引着世界上众多专家、学者为之奋斗。在世界范围内再次掀起了神经网络的研究热潮。难怪有关国际权威人士评论指出，目前对神经网络研究的重要意义不亚于第二次世界大战时对原子弹的研究。

人工神经网络特有的非线性适应性信息处理能力，克服了传统人工智能方法对于直觉（如模式、语音识别、非结构化信息处理方面）的缺陷，使之在神经专家系统、模式识别、智能控制、组合优化、预测等领域得到成功应用。人工神经网络与其他传统方法相结合，将推动人工智能和信息处理技术不断发展。近年来，人工神经网络正向模拟人类认知的道路上更加深入发展，与模糊系统、遗传算法、进化机制等结合，形成计算智能，成为人工智能的一个重要方向，将在实际应用中得到发展。

使用神经网络的主要优点是能够自适应样本数据，当数据中有噪声、形变和非线性时，它也能够正常地工作，很容易继承现有的领域知识，使用灵活，能够处理来自多个资源

和决策系统的数据：提供简单工具进行自动特征选取，产生有用的数据表示，可作为专家系统的前端（预处理器）。此外，神经网络还能提供十分快的优化过程，尤其以硬件直接实现网络时，而且可以加速联机应用程序的运行速度。当然，过分夸大神经网络的应用能力也是不恰当的，毕竟它不是无所不能的。这就需要在实际工作中具体分析问题，合理选择。

基于神经网络的控制称为神经网络控制（NNC），简称神经控制（Neuro Control，NC）。这一新词是在国际自控联杂志《自动化》（Automatica）1994 年 No.11 首次使用的，最早源于 1992 年 H.Tolle 和 E.Ersu 的专著《Neuro Control》。基于神经网络的智能模拟用于控制，是实现智能控制的一种重要形式，近年来获得了迅速发展。本章介绍神经网络的基本概念、基本结构、神经控制系统的组成及实现神经控制的基本方法。

1.1 神经网络的基本概念

1.1.1 生物神经元的结构与功能特点

神经生理学和神经解剖学证明了人的思维是由人脑完成的。神经元是组成人脑的最基本单元，它能够接收并处理信息，人脑大约由 10^{11}～10^{12} 个神经元组成，其中每个神经元约与 10^4～10^5 个神经元通过突触连接。因此，人脑是一个复杂的信息并行加工处理巨系统。探索脑组织的结构、工作原理及信息处理的机制，是整个人类面临的一项挑战，也是整个自然科学的前沿领域。

1．生物神经元的结构

生物神经元，也称神经细胞，是构成神经系统的基本单元。生物神经元主要由细胞体、树突和轴突构成，其基本结构如图 1-1 所示。

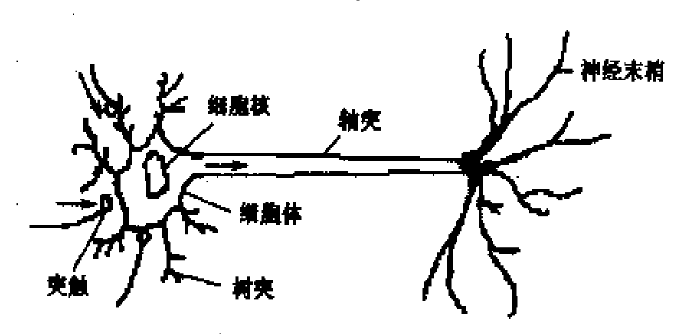


图 1-1 生物神经元结构

1）细胞体

细胞体由细胞核、细胞质、细胞膜等组成。它的直径为 5～100μm，大小不等。细胞体是生物神经元的主体，它是生物神经元的新陈代谢中心，同时还负责接收并处理从其他生物神经元传递过来的信息。细胞体的内部是细胞核，外部是细胞膜。细胞膜外是许多外延的纤维，细胞膜内外有电位差，称为膜电位。膜外为正，膜内为负。

2）轴突

轴突是由细胞体向外伸出的所有纤维中最长的一条分支。每个生物神经元只有一个轴突，长度最大可达 1m 以上，其作用相当于生物神经元的输出电缆，它通过尾部分出的许多神经末梢以及梢端的突触向其他生物神经元输出神经冲动。

3）树突

树突是由细胞体向外伸出的除轴突外的其他纤维分支，长度一般均较短，但分支很多。它相当于生物神经元的输入端，用于接收从四面八方传来的神经冲动。

4）突触

突触是轴突的终端，是生物神经元之间的连接接口，每一个生物神经元约有 10^4～10^5 个突触。一个生物神经元通过其轴突的神经末梢，经突触与另一生物神经元的树突连接，以实现信息的传递。

2. 生物神经元的功能特点

从生物控制论的观点来看，作为控制和信息处理基本单元的生物神经元，具有以下功能特点。

1）时空整合功能

生物神经元对于不同时间通过同一突触传入的信息，具有时间整合功能；对于同一时间通过不同突触传入的信息，具有空间整合功能。两种功能相互结合，使生物神经元具有时空整合的输入信息处理功能。

2）动态极化性

在每一种生物神经元中，信息都是以预知的确定方向流动的，即从生物神经元的接收信息部分（细胞体、树突）传到轴突的起始部分，再传到轴突终端的突触，最后再传给另一生物神经元。尽管不同的生物神经元在形状及功能上都有明显的不同，但大多数生物神经元都是按这一方向进行信息流动的。

3）兴奋与抑制状态

生物神经元具有两种常规工作状态，即兴奋状态与抑制状态。所谓兴奋状态是指生物神经元对输入信息经整合后使细胞膜电位升高，且超过了动作电位的阈值，此时产生神经冲动并由轴突输出。抑制状态是指对输入信息整合后，细胞膜电位值下降到低于动作电位的阈值，从而导致无神经冲动输出。

4）结构的可塑性

突触传递信息的特性是可变的，随着神经冲动传递方式的变化，传递作用强弱不同，会形成生物神经元之间连接的柔性，这种特性又称为生物神经元结构的可塑性。

5）脉冲与电位信号的转换

突触界面具有脉冲与电位信号的转换功能。沿轴突传递的电脉冲是等幅的、离散的脉冲信号，而细胞膜电位变化为连续的电位信号，这两种信号是在突触接口进行变换的。

6）突触延期和不应期

突触对信息的传递具有时延和不应期，在相邻的两次输入之间需要一定的时间间隔，在此期间，无激励，不传递信息，称为不应期。

7）学习、遗忘和疲劳

由于生物神经元结构的可塑性，突触的传递作用有增强、减弱和饱和的情况，因此神经细胞也具有相应的学习、遗忘和疲劳效应（饱和效应）。

1.1.2 人工神经元模型

生物神经元经抽象化后，可得到如图1-2所示的一种人工神经元模型。它有三个基本要素。

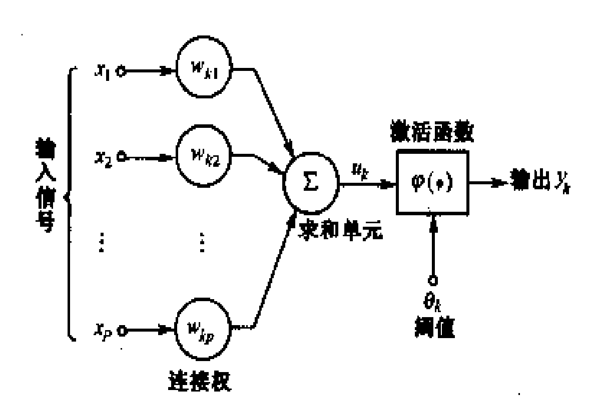


图1-2 人工神经元模型

1. 连接权

连接权对应于生物神经元的突触，各个人工神经元之间的连接强度由连接权的权值表示，权值为正表示激活，为负表示抑制。

2. 求和单元

求和单元用于求取各输入信号的加权和（线性组合）。

3. 激活函数

激活函数起非线性映射作用，并将人工神经元输出幅度限制在一定范围内，一般限制在（0,1）或（−1,1）之间。激活函数也称传输函数。

此外，还有一个阈值θ_k（或偏值$b_k=-\theta_k$）。

以上作用可分别用数学式表达出来，即

$$u_k=\sum_{j=1}^{p}w_{kj}x_j,$$

$$v_k=\text{net}_k=u_k-\theta_k,$$

$$y_k=\varphi(v_k)$$

式中，$x_1,x_2,\cdots,x_p$为输入信号，它相当于生物神经元的树突，为人工神经元的输入信息；

$w_{k1}, w_{k2}, \cdots, w_{kp}$ 为神经元 k 的权值；u_k 为线性组合结果；θ_k 为阈值；$\varphi(\cdot)$ 为激活函数；y_k 为神经元 k 的输出，它相当于生物神经元的轴突，为人工神经元的输出信息。

若把输入的维数增加一维，则可把阈值 θ_k 包括进去，即

$$u_k = \sum_{j=0}^{p} w_{kj} x_j, \quad y_k = \varphi(u_k)$$

此处增加了一个新的连接，其输入 $x_0 = \mp 1$，权值 $w_{k0} = \theta_k$（或 b_k），如图 1-3 所示。

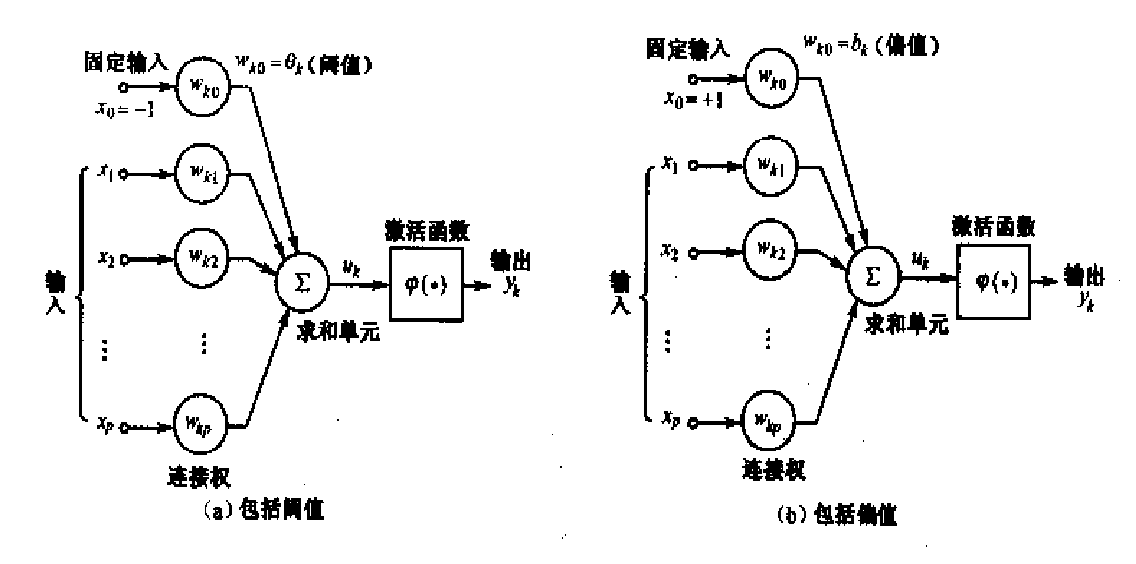


图 1-3　输入扩维后的人工神经元模型

激活函数 $\varphi(\bullet)$ 一般有以下几种形式。

1）阶跃函数

函数表达式为

$$y = \varphi(x) = \begin{cases} 1 & x \geqslant 0 \\ -1 & x < 0 \end{cases}$$

2）分段线性函数

函数表达式为

$$y = \varphi(x) = \begin{cases} 1 & (x \geqslant 1) \\ \frac{1}{2}(1+x) & (-1 < x < 1) \\ -1 & (x \leqslant -1) \end{cases}$$

3）Sigmoid 型函数

最常用的 Sigmoid 型函数为

$$\varphi(x) = \frac{1}{1+\exp(-ax)}$$

式中，参数 a 可控制其斜率。

Sigmoid型函数也简称为S型函数，上式表示的是一种非对称S型函数。

另一种常用的Sigmoid型函数为双曲正切对称S型函数，即

$$\varphi(x)=\tanh\left(\frac{1}{2}x\right)=\frac{1-\exp(-x)}{1+\exp(-x)}$$

这类函数具有平滑和渐近线，并保持单调性。

1.1.3 神经网络的结构

人工神经网络（Artificial Neural Networks，ANN）是由大量人工神经元经广泛互连而组成的，它可用来模拟脑神经系统的结构和功能。人工神经网络可以看成是以人工神经元为节点，用有向加权弧连接起来的有向图。在此有向图中，人工神经元（以下在不易引起混淆的情况下，人工神经元简称神经元）就是对生物神经元的模拟，而有向加权弧则是轴突—突触—树突对的模拟。有向弧的权值表示相互连接的两个人工神经元间相互作用的强弱。

人工神经网络是生物神经网络的一种模拟和近似。它主要从两个方面进行模拟。一种是从生理结构和实现机理方面进行模拟，它涉及生物学、生理学、心理学、物理及化学等许多基础科学。生物神经网络的结构和机理相当复杂，现在距离完全认识它们还相差甚远。另外一种是从功能上加以模拟，即尽量使得人工神经网络具有生物神经网络的某些功能特性，如学习、识别、控制等功能。本书仅讨论后者，从功能上来看，人工神经网络（以下简称神经网络，NN）根据连接方式主要分为两类。

1. 前馈型网络

前馈神经网络是整个神经网络体系中最常见的一种网络，其网络中各个神经元接收前一级的输入，并输出到下一级，网络中没有反馈，如图1-4所示。节点分为两类，即输入单元和计算单元，每一计算单元可有任意个输入，但只有一个输出（它可耦合到任意多个其他节点作为输入）。通常前馈网络可分为不同的层，第 i 层的输入只与第 $i-1$ 层输出相连，输入和输出节点与外界相连，而其他中间层称为隐含层，它们是一种强有力的学习系统，其结构简单而易于编程。从系统的观点看，前馈神经网络是一静态非线性映射，通过简单非线性处理的复合映射可获得复杂的非线性处理能力。但从计算的观点看，前馈神经网络并非是一种强有力的计算系统，不具有丰富的动力学行为。大部分前馈神经网络是学习网络，并不注意系统的动力学行为，它们的分类能力和模式识别能力一般强于其他

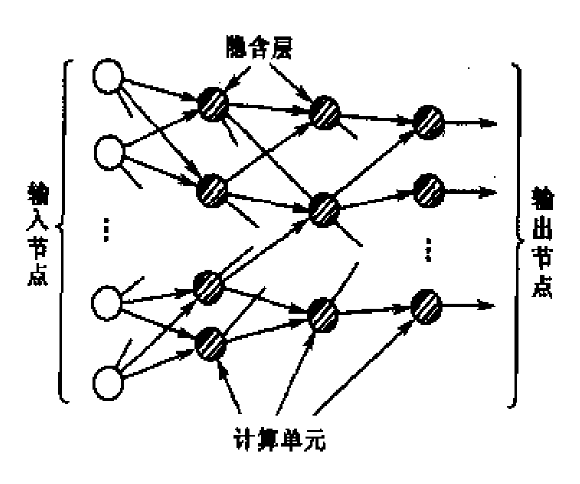


图1-4 前馈网络

类型的神经网络。

2．反馈型网络

反馈神经网络又称递归网络或回归网络。在反馈网络中（Feedback NNs），输入信号决定反馈系统的初始状态，然后系统经过一系列状态转移后，逐渐收敛于平衡状态。这样的平衡状态就是反馈网络经计算后输出的结果，由此可见，稳定性是反馈网络中最重要的问题之一。如果能找到网络的 Lyapunov 函数，则能保证网络从任意的初始状态都能收敛到局部最小点。反馈神经网络中所有节点都是计算单元，同时也可接收输入，并向外界输出，可画成一个无向图，如图 1-5（a）所示，其中每个连接弧都是双向的，也可画成如图 1-5（b）所示的形式。若总单元数为 n，则每一个节点有 $n-1$ 个输入和一个输出。

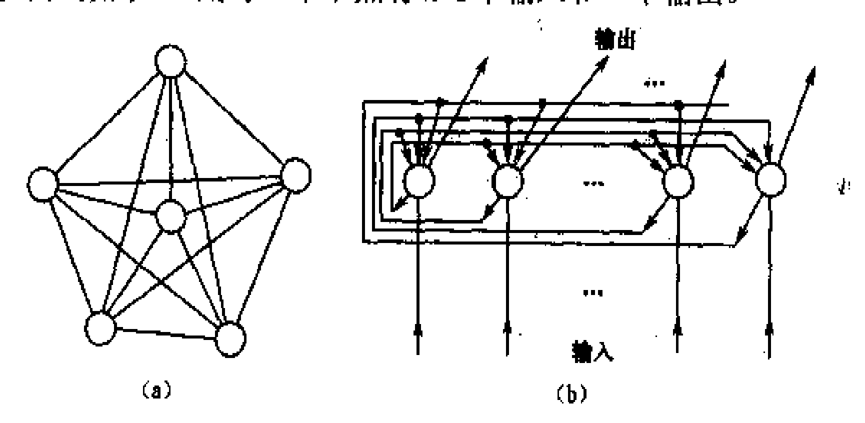


图 1-5　单层全连接反馈网络

1.1.4　神经网络的工作方式

神经网络的工作过程主要分为两个阶段：第一阶段是学习期，此时各计算单元状态不变，各连接权上的权值可通过学习来修改；第二阶段是工作期，此时各连接权固定，计算单元变化，以达到某种稳定状态。

从作用效果看，前馈网络主要是函数映射，可用于模式识别和函数逼近。反馈网络按对能量函数的极小点的利用来分类有两种：第一类是能量函数的所有极小点都起作用，这一类主要用做各种联想存储器；第二类只利用全局极小点，它主要用于求解最优化问题。

1.1.5　神经网络的学习

1．学习方式

通过向环境学习获取知识并改进自身性能是神经网络的一个重要特点，在一般情况下，性能的改善是按某种预定的度量调节自身参数（如权值）并随时间逐步达到的，学习方式（按环境所供信息的多少分）有以下三种。

1）有监督学习（有教师学习）

有监督学习方式需要外界存在一个“教师”，它可对一组给定输入提供应有的输出结果

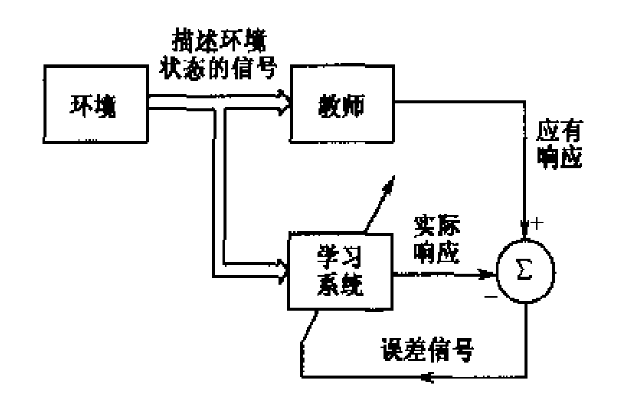


图1-6 有监督学习框图

（正确答案），这组已知的输入/输出数据称为训练样本集。学习系统可根据已知输出与实际输出之间的差值（误差信号）来调节系统参数，如图1-6所示。在有监督学习当中，学习规则由一组描述网络行为的训练集给出

$$\{x_1, t_1\},\{x_2, t_2\},\cdots,\{x_N, t_N\}$$

式中，x_i为网络的输入；t_i为相应的目标输出。当输入作用到网络时，网络的实际输出与目标输出相比较，然后学习规则调整网络的权值和阈值，从而使网络的实际输出越来越接近于目标输出。

2）无监督学习（无教师学习）

无监督学习时不存在外部教师，学习系统完全按照环境所提供数据的某些统计规律来调节自身参数或结构（这是一种自组织过程），以表示外部输入的某种固有特性（如聚类，或某种统计上的分布特征），如图1-7所示。在无监督学习当中，仅仅根据网络的输入调整网络的权值和阈值，它没有目标输出。乍一看，这种学习似乎并不可行：不知道网络的目的是什么，还能够训练网络吗？实际上，大多数这种类型的算法都是要完成某种聚类操作，学会将输入模式分为有限的几种类型。这种功能特别适合于诸如向量量化等应用问题。

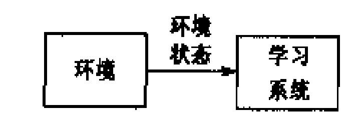


图1-7 无监督学习框图

3）强化学习（或再励学习）

强化学习介于上述两种情况之间，外部环境对系统输出结果只给出评价（奖或罚）而不是给出正确答案，学习系统通过强化那些受奖励的动作来改善自身性能，如图1-8所示。强化学习与有监督学习类似，只是它不像有监督的学习那样为每一个输入提供相应的目标输出，而是仅仅给出一个级别。这个级别（或评分）是对网络在某些输入序列上的性能测度。当前，这种类型的学习要比有监督的学习少见。它最适合控制系统应用领域。

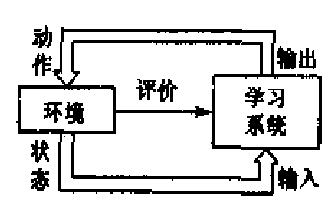


图1-8 强化学习框图

2. 学习算法

1）δ学习规则（误差纠正规则）

若$y_i(k)$为输入$x(k)$时神经元i在k时刻的实际输出，$t_i(k)$表示相应的期望输出，则误差信号可写为

$$e_i(k) = t_i(k) - y_i(k)$$

误差纠正学习的最终目的是使某一基于$e_i(k)$的目标函数达最小，以使网络中每一输出单元的实际输出在某种统计意义上最逼近于期望输出。一旦选定了目标函数形式，误差纠正

学习就成为一个典型的最优化问题。最常用的目标函数是均方误差判据，定义为

$$J=E\left\{\frac{1}{2}\sum_{i=1}^{N}(t_i-y_i)^2\right\}$$

式中，E 是统计期望算子，上式的前提是被学习的过程是宽而平稳的，具体方法可用最陡梯度下降法。直接用 J 作为目标函数时，需要知道整个过程的统计特性，为解决这一困难用 J 在时刻 k 的瞬时值 $J(k)$代替 J，即

$$J(k)=\frac{1}{2}\sum_{i=1}^{N}(t_i-y_i)^2=\frac{1}{2}\sum_{i=1}^{N}e_i^2(k)$$

问题变为求 $J(k)$对权值 w_{ij}的极小值，根据最陡梯度下降法可得

$$\Delta w_{ij}(k)=\eta\cdot\delta_i(k)\cdot x_j(k)=\eta\cdot e_i(k)\cdot f'(W_i x)\cdot x_j(k)$$

式中，η 为学习速率或步长（$0<\eta\leqslant 1$），$f(\bullet)$为激活函数。这就是通常说的误差纠正学习规则（或称δ规则），用于控制每次误差修正值。它是基于使输出方差最小的思想而建立的。

2）Hebb 学习规则

神经心理学家 Hebb 提出的学习规则可归结为“当某一突触（连接）两端的神经元激活同步（同为激活或同为抑制）时，该连接的强度应增加，反之则应减弱”，用数学方式可描述为

$$\Delta w_{ij}(k)=F(y_i(k),x_j(k))$$

式中，$y_i(k), x_j(k)$分别为 w_{ij}两端神经元的状态，其中最常用的一种情况为

$$\Delta w_{ij}(k)=\eta\cdot y_i(k)\cdot x_j(k)$$

式中，η 为学习速率。由于 $w_{ij}(k)$与 $y_i(k)$、$x_j(k)$的相关成比例，故有时称之为相关学习规则。上式定义的 Hebb 规则实际上是一种无监督的学习规则，它不需要关于目标输出的任何相关信息。

原始的 Hebb 学习规则对权值矩阵的取值未做任何限制，因而学习后权值可取任意值。为了克服这一弊病，在 Hebb 学习规则的基础上增加一个衰减项，即

$$\Delta w_{ij}(k)=\eta\cdot y_i(k)\cdot x_j(k)-d_r * w_{ij}(k)$$

衰减项的加入能够增加网络学习的“记忆”功能，并且能够有效地对权值的取值加以限制。衰减系数 d_r的取值在[0,1]之间。当取 0 时，就变成原始的 Hebb 学习规则。

另外，Hebb 学习规则还可以采用有监督的学习，对于有监督学习的 Hebb 学习规则而言，是将目标输出代替实际输出。由此，算法被告知的就是网络应该做什么，而不是网络当前正在做什么，可描述为

$$\Delta w_{ij}(k)=\eta\cdot t_i(k)\cdot x_j(k)$$

3）竞争（Competitive）学习

顾名思义，在竞争学习时网络各输出单元互相竞争，最后达到只有一个最强者激活。

最常见的一种情况是输出神经元之间有侧向抑制性连接，如图 1-9 所示。这样众多输出单元中如有某一单元较强，则它将获胜并抑制其他单元，最后只有比较强者处于激活状态。最常用的竞争学习规则有以下三种：

Kohonen 规则：$\Delta w_{ij}(k)=\begin{cases}\eta(x_j-w_{ij})，& 若神经元 j 竞争获胜\\ 0 ，& 若神经元 j 竞争失败\end{cases}$

Instar 规则：$\Delta w_{ij}(k)=\begin{cases}\eta y_i(x_j-w_{ij})，& 若神经元 j 竞争获胜\\ 0 ，& 若神经元 j 竞争失败\end{cases}$

Outstar 规则：$\Delta w_{ij}(k)=\begin{cases}\eta(y_i-w_{ij})/x_j，& 若神经元 j 竞争获胜\\ 0 ，& 若神经元 j 竞争失败\end{cases}$

3. 学习与自适应

当学习系统所处环境平稳时（统计特征不随时间变化），从理论上说通过监督学习可以学到环境的统计特征，这些统计特征可被学习系统（神经网络）作为经验记住。如果环境是非平稳的（统计特征随时间变化），通常的监督学习没有能力跟踪这种变化，为解决此问题需要网络有一定的自适应能力，此时对每一个不同输入都作为一个新的例子对待。其工作过程如图 1-10 所示。此时模型（如 NN）被当作一个预测器，基于前一时刻输出 $x(k-1)$和模型在 $k-1$ 时刻的参数，它估计出 k 时刻的输出 $\hat{x}(k)$，$\hat{x}(k)$ 与实际值 $x(k)$（作为应有的正确答案）比较，其差值 $e(k)$称为“新息”，如新息 $e(k)=0$，则不修正模型参数，否则应修正模型参数，以便跟踪环境的变化。

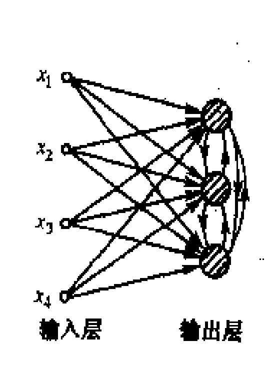


图 1-9 竞争学习网络

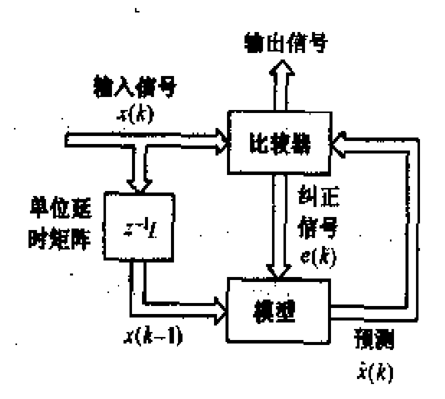


图 1-10 自适应学习框图

1.1.6 神经网络的分类

神经网络根据不同的情况，可按以下几方面进行分类：

（1）按功能分：连续型与离散型、确定型与随机型、静态与动态神经网络；

（2）按连接方式分：前馈（或称前向）型与反馈型神经网络；

（3）按逼近特性分：全局逼近型与局部逼近型神经网络；

（4）按学习方式分：有监督学习、无监督学习和强化学习神经网络。

1.2 典型神经网络的模型

自 1957 年 F. Rosenblatt 在第一届人工智能会议上展示他构造的第一个人工神经网络模型以来，据统计到目前为止已有上百种神经网络问世。根据 HCC 公司及 IEEE 的调查统计，有十多种神经网络比较著名，以下按照神经网络的拓扑结构与学习算法相结合的方法，将神经网络的类型分为前馈网络、竞争网络、反馈网络和随机网络四大类，并按类介绍 MP 模型、感知机神经网络、自适应线性神经网络（Adaline）、BP 神经网络、径向基神经网络、自组织竞争神经网络、自组织特征映射神经网络（SOM）、反传神经网络（CPN）、自适应共振理论（ART）神经网络、学习向量量化（LVQ）神经网络、Elman 神经网络、Hopfield 神经网络和 Boltzmann 神经网络的特点、拓扑结构、工作原理和学习机理，以揭示神经网络所具有的功能和特征。运用这些神经网络模型可实现函数逼近、数据聚类、模式分类、优化计算等功能。因此，神经网络广泛应用于人工智能、自动控制、机器人、统计学等领域的信息处理中。

1.2.1 MP 模型

MP 模型最初是由美国心理学家 McCulloch 和数学家 Pitts 在 1943 年共同提出的，它具有固定的结构和不变的权值，它的权分为兴奋性突触权和抑制性突触权两类，如抑制性突触权被激活，则神经元被抑制，输出为零。而兴奋性突触权的数目比较多，兴奋性突触权能否激活，则要看它的累加值是否大于一个阈值，大于该阈值神经元即兴奋。

MP 模型中单个神经元示意图如图 1-11 所示，变换关系为

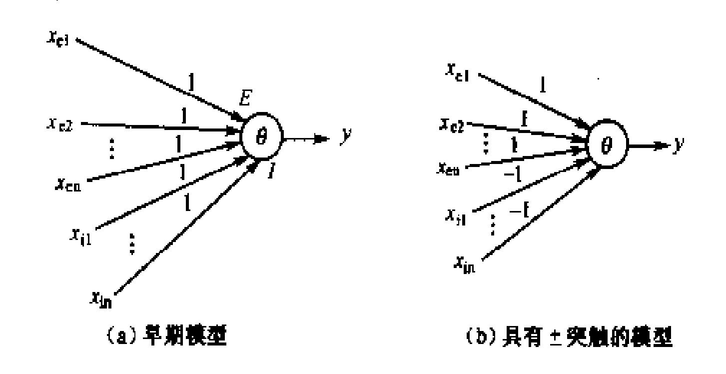
$$E=\sum_{j=1}^{n}x_{ej},\ I=\sum_{k=1}^{n}x_{ik}$$

式中，$x_{ej}(j=1, 2, \cdots, n)$为兴奋性突触的输入；$x_{ik}(k=1, 2, \cdots, n)$为抑制性突触的输入，则输入与输出的转换关系为

$$y=\begin{cases}1 & ,I=0,\ \ E\geqslant\theta \\ 0 & ,I=0,\ \ E<\theta \\ 0 & ,I>0\end{cases}$$

MP 模型是早期提出的。在如图 1-11（a）所示中，模型的权值均为 1，它可以用来完成一些逻辑性关系。如果兴奋与抑制突触用权±1 表示，而总的作用用加权的办法实现，兴奋为 1，抑制为−1，如图 1-11（b）所示，则有

$$y=\begin{cases}1 & \sum_{j=1}^{n}x_{ej}-\sum_{k=1}^{n}x_{ik}\geqslant\theta \\ 0 & \sum_{j=1}^{n}x_{ej}-\sum_{k=1}^{n}x_{ik}<\theta\end{cases}$$



(a) 早期模型 (b) 具有±突触的模型

图 1-11 MP 模型中单个神经元示意图

图 1-12 (a)、(b)、(c)、(d) 和 (e) 是利用 MP 模型分别表示的或、与、非及一些逻辑关系式。

MP 模型的权值、输入和输出都是二值变量，这同由逻辑门组成的逻辑关系式的实现区别不大，又由于它的权值无法调节，因而现在很少有人单独使用。但它是人工神经元模型的基础，也是神经网络理论的基础。

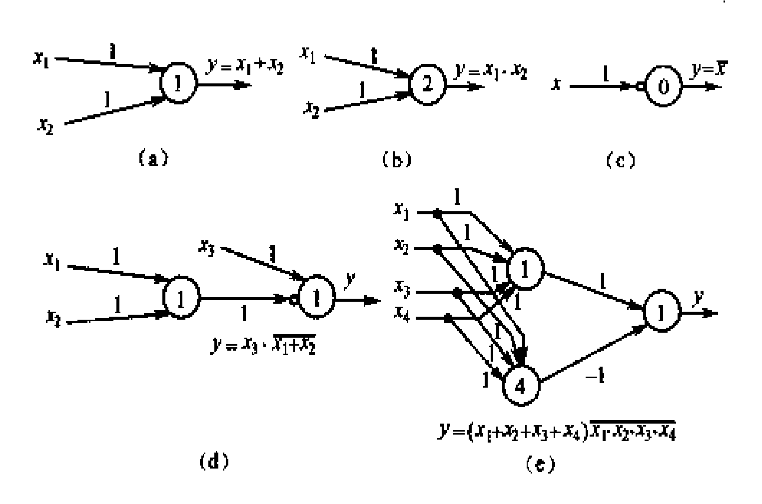


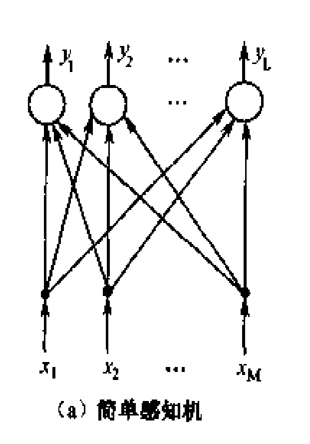
图 1-12 用 MP 模型实现的布尔逻辑

1.2.2 感知机神经网络

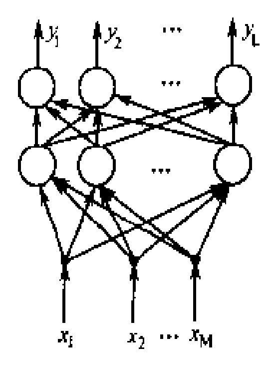
1. 感知机的网络结构

1957 年，美国心理学家 Frank Rosenblatt 及其合作者为了研究大脑的存储、学习和认知过程而提出了一类神经网络模型，并称其为感知机（Perceptron）。感知机较 MP 模型又进了一步，它的输入可以是非离散量，它的权值不仅是非离散量，而且可以通过调整学习而得到。感知机可以对输入的样本矢量进行模式分类，而且多层的感知机，在某些样本点上对函数进行逼近，但感知机是一个线性阈值单元组成的网络，在结构和算法上是其他前馈网络的基础，尤其它对隐单元的选取比其他非线性阈值单元组成的网络容易分析，而对感知机的讨论，可以对其他网络的分析提供依据。由于感知机的权值可以通过学习调整而得到，因此它

被认为是最早提出的一种神经网络模型。图 1-13 为感知机的两种结构。



(a) 简单感知机



(b) 两层感知机

图 1-13 感知机的两种结构

在这种模型中，输入模式 $\boldsymbol{x}=[x_1, x_2, \cdots, x_M]^{\mathrm{T}}$ 通过各输入端点分配给下一层的各节点，下一层可以是中间层（或称为稳含层），也可以是输出层，隐含层可以是一层也可以是多层，最后通过输出层节点得到输出模式 $\boldsymbol{y}=[y_1, y_2, \cdots, y_L]^{\mathrm{T}}$。在这类前馈网络中没有层内连接，也没有隔层的前馈连接。每一节点只能前馈连接到其下一层的所有节点。然而，对于含有隐含层的多层感知机，当时没有可行的训练方法，初期研究的感知机为一层感知机或称为简单感知机，通常就把它称为感知机。虽然简单感知机有其局限性，但人们对它进行了深入的研究，有关它的理论仍是研究其他网络模型的基础。

如果在输入层和输出层单元之间加入一层或多层处理单元，即可构成多层感知机，因而多层感知机由输入层、隐含层、输出层组成。隐含层的作用相当于特征检测器，提取输入模式中包含的有效特征信息，使输出单元所处理的模式是线性可分的。但需注意，多层感知机模型只允许一层连接权值可调，原因是无法设计出一个有效的多层感知机学习算法。图 1-14 是一个两层感知机结构（包括输入层、一个隐含层和一个输出层），有两层连接权，其中输入层和隐含层单元间的连接权值是随机设定的固定值，不可调节；隐含层与输出层单元间的连接权值是可调的。

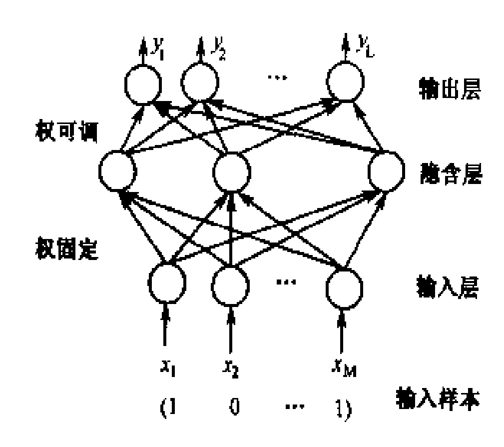


图 1-14 两层感知机的结构

值得注意的是，在神经网络中，由于输入层仅仅起输入信号的等值传输作用，而不对信号进行运算，故在定义多少层神经网络时，一般不把输入层计算在内，如上所述。也就是说，一般把隐含层称为神经网络的第一层，输出层称为神经网络的第二层（假如只有一个隐含层）。如果有两个隐含层，则第一个隐含层称为神经网络的第一层，第二个隐含层称为神经网络的第二层，而输出层称为神经网络的第三层。如果有多个隐含层，则依此类推。在

MATLAB 神经网络工具箱中的定义也类同。

感知机神经网络是由 hardlim 产生的符号函数阈值元件组成的。对于具有 M 个输入、L 个输出的单层感知机网络，如图 1-13（a）所示。该网络通过一组权值 w_{ij}(i=1, 2, …, L；j=1, 2, …, M)与 L 个神经元组成。根据结构图，可以写出，输出层的第 i 个神经元的输入总和和输出分别为

$$\text{net}_i = \sum_{j=1}^{M} w_{ij} x_j - \theta_i, y_i = f(\text{net}_i) \qquad (i=1,\ 2,\ \cdots,\ L) \tag{1-1}$$

式中，θ_i 为输出层神经元 i 的阈值；M 为输入层的节点数，即输入的个数；$f(\bullet)$ 为激活函数。感知机中的激活函数使用了阶跃限幅函数，因此感知机能够将输入向量分为两个区域，即

$$f(x) = \begin{cases} 1, x \geqslant 0 \\ 0, x < 0 \end{cases}$$

2．感知机学习

感知机的学习是典型的有教师学习，可以通过样本训练达到学习的目的。训练的条件有两个，即训练集和训练规则。感知机的训练集就是由若干个输入/输出模式对构成的一个集合，所谓输入/输出模式对是指由一个输入模式及其期望输出模式所组成的向量对。它包括二进制值输入模式及其期望输出模式，每个输出对应一个分类。F. Rosenblatt 业已证明，如果两类模式是线性可分的（指存在一个超平面将它们分开），则算法一定收敛。

设有 N 个训练样本，在感知机训练期间，不断用训练集中的每个模式对训练网络。当给定某一个样本 p 的输入/输出模式对时，感知机输出单元会产生一个实际输出向量，用期望输出与实际的输出之差来修正网络连接权值。权值的修正采用简单的误差学习规则（即δ规则），它是一个有教师的学习过程，其基本思想是利用某个神经单元的期望输出与实际的输出之间的差来调整该神经单元与上一层中相应神经单元的连接权值，最终减小这种偏差。也就是说，神经单元之间连接权的变化正比于输出单元期望输出与实际的输出之差。

简单感知机输出层的任意神经元 i 的连接权值 w_{ij} 和阈值 θ_i 的修正公式为

$$\Delta w_{ij} = \eta(t_i^p - y_i^p) \cdot x_j^p = \eta e_i^p \cdot x_j^p \qquad (i=1, 2, \cdots, L;\ j=1, 2, \cdots, M) \tag{1-2}$$

$$\Delta \theta_i = \eta(t_i^p - y_i^p) \cdot 1 = \eta e_i^p \qquad (i=1,\ 2,\ \cdots,\ L) \tag{1-3}$$

式中，t_i^p 表示在样本 p 作用下的第 i 个神经元的期望输出；y_i^p 表示在样本 p 作用下的第 i 个神经元的实际输出；η 为学习速率（$0<\eta\leqslant1$），用于控制权值调整速度。学习速率 η 较大时，学习过程加速，网络收敛较快。但是 η 太大时，学习过程变得不稳定，且误差会加大。因此学习速率的取值很关键。

感知机的学习规则属误差纠正规则，该法已被证明，经过若干次迭代计算后，可以收敛到正确的目标向量。由上可知，该算法无需求导数，因此比较简单，又具有收敛速度快和精度高的优点。

期望输出与实际输出之差为

$$e_i^p = t_i^p - y_i^p = \begin{cases} 1 & (t_i^p = 1, y_i^p = 0) \\ 0 & (t_i^p = y_i^p) \\ -1 & (t_i^p = 0, y_i^p = 1) \end{cases} \tag{1-4}$$

由此可见，权值变化量与两个量有关，即输入状态 x_j 和输出误差。当且仅当输出单元 i 有输出误差且相连输入状态 x_j 为1时，修正权值或增加一个量或减少一个量。

感知机的学习过程又称为最小方差学习过程。根据权向量分布，可以构造一个多维权空间，其中，每个权对应一个轴，另一个轴表示学习过程中的误差度量。对每个权向量都会有一定输出误差，由权空间某点的“高度”表示。学习过程中所有这些点形成的一个空间表面，称为误差表面。线性输出单元的感知机，其误差表面成一碗形，其水平截线为椭圆，垂直截线为抛物线。显然，该碗形表面只有一个极小点，沿误差表面按梯度下降法就能达到该点，这涉及感知机学习的收敛性，下面还要详细讨论。

3. 感知机的线性可分性

感知机可以对线性可分性输入模式进行分类，如两维输入 x_1, x_2。其分界线为 n–1 维（2–1=1）直线，则 $w_1x_1+w_2x_2-\theta=0$

根据式（1-1）可知，当且仅当 $w_1x_1+w_2x_2 \geqslant \theta$ 时，y=1，此时把输入模式划分为“1”类，用“•”代表输出为1的输入模式，即目标输出为1的两个输入向量用黑心圆圈“•”表示；当且仅当 $w_1x_1+w_2x_2<\theta$ 时，y=0，此时把输入模式划分为“0”类，用“o”代表输出为0的输入模式，即目标输出为0的两个输入向量用空心圆圈“o”表示，其对应的线性分割如图1-15所示。感知机对与、或、非问题均可以线性分割。感知机模式只能对线性输入模式进行分类，这是它的主要功能局限。

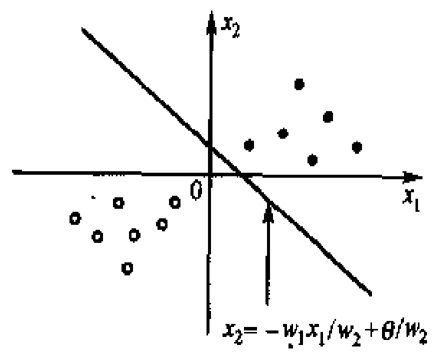


图1-15　线性分割图

例1-1　利用简单感知机对“与”、“或”和“异或”问题进行分类。

解：逻辑“与”、“或” 和“异或”的真值表见表1-1。

表1-1　真值表

x_1	x_2	y		
		AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

对于逻辑“与”和逻辑“或”，可将输入模式按照其输出分成两类：输出为 0 的属于“0”类，用“o”代表；输出为 1 的属于“1”类，用“•”代表，如图 1-16（a）、（b）所示。输入模式可以用一条决策直线划分为两类，即逻辑“与”和逻辑“或”是线性可分的。因此简单感知机可以解决逻辑“与”和逻辑“或”的问题。

对于逻辑“异或”，现仍然将输入模式按照其输出分成两类，即这四个输入模式分布在二维空间中，如图 1-16（c）所示。显然，无法用一条决策直线把这四个输入模式分成两类，即逻辑“异或”是线性不可分的。因此简单感知机无法解决逻辑“异或”问题。

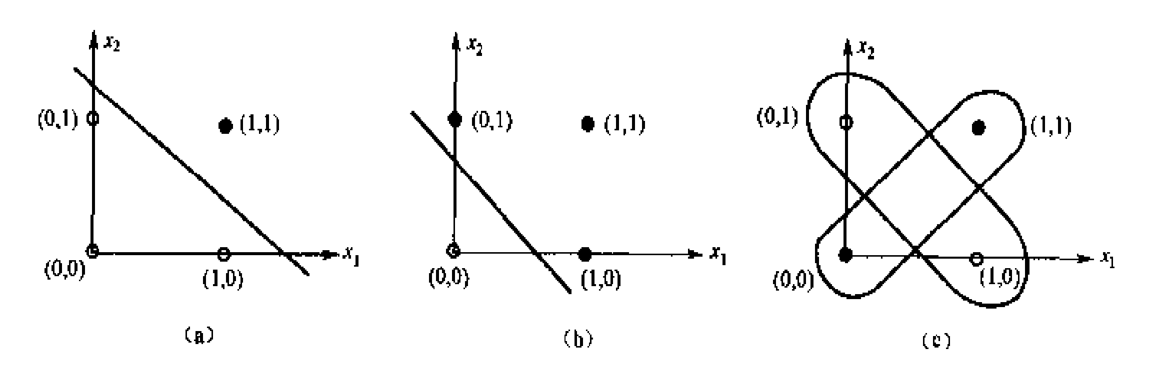


图 1-16 AND、OR 和 XOR 输入模式的空间分布

感知机为解决逻辑异或问题，可以设计一个多层的网络，即含有输入层，隐含层和输出层的结构。

可以证明，只要隐含层单元数足够多，用多层感知机网络可实现任何模型分类。但是，隐含层单元的状态不受外界直接控制，这给多层网络的学习带来极大困难。

4. 感知机收敛性定理

定理 1.1 如果样本输入函数是线性可分的，那么感知机学习算法经过有限次迭代后可收敛到正确的权值或权向量。

定理 1.2 假定隐含层单元可以根据需要自由设置，那么用双隐含层感知机可以实现任意的二值逻辑函数。

5. 感知机网络学习算法的计算步骤

（1）初始化：置所有的加权系数为最小的随机数；

（2）提供训练集：给出顺序赋值的输入向量 $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N$ 和期望的输出向量 $\boldsymbol{t}_1, \boldsymbol{t}_2, \cdots, \boldsymbol{t}_N$；

（3）计算实际输出：按式（1-1）计算输出层各神经元的输出；

（4）按式（1-4）计算期望值与实际输出的误差；

（5）按式（1-2）和式（1-3）调整输出层的加权系数 w_{ij} 和阈值 θ_i；

（6）返回计算（3）步，直到误差满足要求为止。

例 1-2 建立一个感知机网络，使其能够完成“与”的功能。

解：为了完成"与"函数，建立一个两输入、单输出的感知机网络。根据表 1-1 中"与"函数的真值表，可得训练集的输入矩阵为：X=[0 0 1 1;0 1 0 1]，目标向量为：T=[0 0 0 1]。根据感知机网络学习算法的计算步骤，利用 MATLAB 语言编写的程序如下：

```
%感知机神经网络的第一阶段学习期（训练加权系数 Wij）
err_goal=0.001;lr=0.9;                 %给定期望误差最小值和学习速率
max_epoch=10000;                       %给定训练的最大次数
X=[0 0 1 1;0 1 0 1];T=[0 0 0 1];       %提供四组训练集 （2 输入，1 输出）
%初始化 Wij（M 为输入节点 j 的数量,L 为输出节点 i 的数量,N 为为训练集对数量）
[M,N]=size(X);[L,N]=size(T);
Wij=rand(L,M);y=0;b=rand(L);
for epoch=1:max_epoch
%计算各神经元输出
NETi=Wij*X;
for j=1:N
  for i=1:L
    if (NETi(i,j)>=b(i))
        y(i,j)=1;
    else
        y(i,j)=0;
    end
  end
end
%计算误差函数
E=(T-y);EE=0;
for j=1:N
 EE=EE+abs(E(j));
end
if (EE<err_goal) break;end
Wij=Wij+lr*E*X';                       %调整输出层加权系数
b=b+sqrt(EE);                          %调整输出层神经元的阈值
end
epoch,Wij,b                            %显示计算次数、加权系数和阈值
%感知机神经网络的第二阶段工作期（根据训练好的 Wij 和给定的输入计算输出）
X1=X;                                  %给定输入
%计算各神经元输出
```

```
NETi=Wij*X1; [M,N]=size(X1);
for j=1:N
  for i=1:L
    if (NETi(i,j)>=b(i))
        y(i,j)=1;
    else
        y(i,j)=0;
    end
  end
end
y                                       %显示网络的输出
```

其结果显示为

```
Wij =
   2.6462    2.3252
b =
   4.6169
y=
   0   0   0   1
```

1.2.3 自适应线性神经网络

线性神经网络是一种简单的神经元网络，它可以由一个或多个线性神经元构成。1962年由美国斯坦福大学教授 Berhard Widrow 提出的自适应线性元件网络（Adaptive Linear Element，Adaline）是线性神经网络最早的典型代表，它是一个由输入层和输出层构成的单层前馈型网络。它与感知机神经网络的不同之处在于其每个神经元的传输函数为线性函数，因此自适应线性神经网络的输出可以取任意值，而感知机神经网络的输出只能是 1 或 0。线性神经网络采用由 Berhard Widrow 和 Marcian Hoff 共同提出的一种新的学习规则，也称为 Widrow-Hoff 学习规则，或者 LMS（Least Mean Square）算法来调整网络的权值和阈值。自适应线性神经网络的学习算法比感知机网络的学习算法的收敛速度和精度都有较大的提高。自适应线性神经网络主要用于函数逼近、信号预测、系统辨识、模式识别和控制等领域。

1. 自适应线性神经网络结构

自适应线性神经网络结构同感知机，不同之处在于其每个神经元的传输函数为线性函数。对于具有 M 个输入、L 个输出的线性神经网络，输出层的第 i 个神经元的输入总和（即激活函数）和输出分别为

$$\text{net}_i=\sum_{j=1}^{M}w_{ij}x_j-\theta_i,\ y_i=f(\text{net}_i)\qquad (i=1,\ 2,\ \cdots,\ L)\tag{1-5}$$

式中，θ_i 为输出层神经元 i 的阈值；M 为输入层的节点数，即输入的个数；$f(\bullet)$ 为激活函数，它为线性函数的传输函数。

2. 自适应线性神经网络的学习

自适应线性神经网络的学习也是典型的有导师学习，采用 Widrow-Hoff 学习规则，即 LMS 学习规则。在训练期间，不断用训练集中的每个模式对训练网络。当给定某一训练模式时，输出单元会产生一个实际输出向量，用期望输出与实际的输出之差来修正网络连接权值。

在训练网络的学习阶段，设有 N 个训练样本，先假定用其中的某一个样本 p 的输入输出模式对$\{\boldsymbol{X}_\text{p}\}$和$\{\boldsymbol{T}_\text{p}\}$对网络进行训练，输出层的第 i 个神经元在样本 p 作用下的输入为

$$\text{net}_i^p=\sum_{j=1}^{M}w_{ij}x_j^p-\theta_i\qquad (i=1,2,\ \cdots,L)$$

式中，θ_i 为输出层神经元 i 的阈值；M 为输入层的节点数，即输入的个数。

输出层第 i 个神经元的输出为

$$y_i^p=f(\text{net}_i^p)\qquad (i=1,2,\ \cdots,L)$$

式中，$f(\bullet)$ 为线性激活函数。它将网络的输入原封不动地输出，因此有

$$y_i^p=\text{net}_i^p=\sum_{j=1}^{M}w_{ij}x_j^p-\theta_i\qquad (i=1,2,\cdots,L)\tag{1-6}$$

对于每一样本 p 的输入模式对的二次型误差函数为

$$J_p=\frac{1}{2}\sum_{i=1}^{L}(t_i^p-y_i^p)^2=\frac{1}{2}\sum_{i=1}^{L}\text{e}_i^2\tag{1-7}$$

则系统对所有 N 个训练样本的总误差函数为

$$J=\sum_{p=1}^{N}J_p=\frac{1}{2}\sum_{p=1}^{N}\sum_{i=1}^{L}(t_i^p-y_i^p)^2=\frac{1}{2}\sum_{p=1}^{N}\sum_{i=1}^{L}\text{e}_i^2\tag{1-8}$$

式中，N 为模式样本对数；L 为网络输出节点数；t_i^p 表示在样本 p 作用下的第 i 个神经元的期望输出；y_i^p 表示在样本 p 作用下的第 i 个神经元的实际输出。

自适应线性神经网络加权系数修正采用 Widrow-Hoff 学习规则，又称为最小 Square 均方误差算法（LMS）。它的实质是利用梯度最速下降法，是权值沿误差函数的负梯度方向改变。Widrow-Hoff 学习规则的权值变化量正比于网络的输出误差及网络的输入矢量。根据梯度法，可得输出层的任意神经元 i 的加权系数修正公式为

$$\Delta w_{ij}=-\eta\frac{\partial J_p}{\partial w_{ij}}\qquad (i=1,2,\ \cdots,L;\ j=1,2,\ \cdots,M)$$

式中，学习速率$\eta=\frac{\alpha}{\|X_p\|_2^2}$，$\alpha$为常值，当$0<\alpha<2$时，可使算法收敛。$\eta$随着输入样本$X_p$自适应地调整。

因为

$$\frac{\partial J_p}{\partial w_{ij}}=\frac{\partial J_p}{\partial \mathrm{net}_i^p}\cdot\frac{\partial \mathrm{net}_i^p}{\partial w_{ij}}=\frac{\partial J_p}{\partial \mathrm{net}_i^p}\cdot x_j^p$$

定义 $$\delta_i^p=-\frac{\partial J_p}{\partial net_i^p}=-\frac{\partial J_p}{\partial y_i^p}\cdot\frac{\partial y_i^p}{\partial \mathrm{net}_i^p}=(t_i^p-y_i^p)\cdot f'(\mathrm{net}_i^p)=\mathrm{e}_i\cdot f'(\mathrm{net}_i^p)$$

则

$$\Delta w_{ij}=\eta\cdot\delta_i^p\cdot x_j^p$$

由于激活函数$f(\bullet)$为线性函数，故 $f'(\mathrm{net}_i^p)=1$

所以

$$\frac{\partial J_p}{\partial w_{ij}}=-\mathrm{e}_i\cdot x_j^p$$

输出层的任意神经元i的加权系数修正公式为

$$\Delta w_{ij}=\eta\cdot(t_i^p-y_i^p)\cdot x_j^p=\eta\cdot\mathrm{e}_i\cdot x_j^p \tag{1-9}$$

同理，阈值θ_i的修正公式为

$$\Delta\theta_i=\eta(t_i-y_i)=\eta\mathrm{e}_i \tag{1-10}$$

以上两式构成了最小 Square 均方误差算法（LMS），或 Widrow-Hoff 学习算法，它实际上也是δ 学习规则的一种特例。

3. 自适应线性神经网络学习算法的计算步骤

（1）初始化：置所有的加权系数为最小的随机数；

（2）提供训练集：给出顺序赋值的输入向量$\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N$和期望的输出向量$\boldsymbol{t}_1, \boldsymbol{t}_2, \cdots, \boldsymbol{t}_N$；

（3）计算实际输出：按式（1-5）或式（1-6）计算输出层各神经元的输出；

（4）按式（1-7）或式（1-8）计算期望值与实际输出的误差；

（5）按式（1-9）和式（1-10）调整输出层的加权系数w_{ij}和阈值θ_i；

（6）返回计算（3）步，直到误差满足要求为止。

1.2.4 BP 神经网络

1986 年，D.E.Rumelhart 和 J.L.McClelland 提出了一种利用误差反向传播训练算法的神经网络，简称 BP（Back Propagation）网络，是一种有隐含层的多层前馈网络，系统地解决

了多层网络中隐含单元连接权的学习问题。

如果网络的输入节点数为 M、输出节点数为 L，则此神经网络可看成是从 M 维欧氏空间到 L 维欧氏空间的映射。这种映射是高度非线性的，其主要用于：

（1）模式识别与分类：用于语言、文字、图像的识别，医学特征的分类和诊断等。

（2）函数逼近：用于非线性控制系统的建模、机器人的轨迹控制及其他工业控制等。

（3）数据压缩：用于编码压缩和恢复，图像数据的压缩和存储以及图像特征的抽取等。

1. BP 算法原理

BP 学习算法的基本原理是梯度最速下降法，它的中心思想是调整权值使网络总误差最小。也就是采用梯度搜索技术，以期使网络的实际输出值与期望输出值的误差均方值为最小。网络学习过程是一种误差边向后传播边修正权系数的过程。

多层网络运用 BP 学习算法时，实际上包含了正向和反向传播两个阶段。在正向传播过程中，输入信息从输入层经隐含层逐层处理，并传向输出层，每一层神经元的状态只影响下一层神经元的状态。如果在输出层不能得到期望输出，则转入反向传播，将误差信号沿原来的连接通道返回，通过修改各层神经元的权值，使误差信号最小。

将上一层节点的输出传送到下一层时，通过调整连接权系数 w_{ij} 来达到增强或削弱这些输出的作用。除了输入层的节点外，隐含层和输出层节点的净输入是前一层节点输出的加权和。每个节点的激活程度由它的输入信号、激活函数和节点的偏置（或阈值）来决定。但对于输入层，输入模式送到输入层节点上，这一层节点的输出即等于其输入。注意，这种网络没有反馈存在，实际运行仍是单向的，所以不能将其看成是一非线性动力学系统，而只是一种非线性映射关系。具有隐含层 BP 网络的结构如图 1-17 所示，图中设有 M 个输入节点 $x_1, x_2, \cdots, x_M$，L 个输出节点 $y_1, y_2, \cdots, y_L$，网络的隐含层共有 q 个神经元。

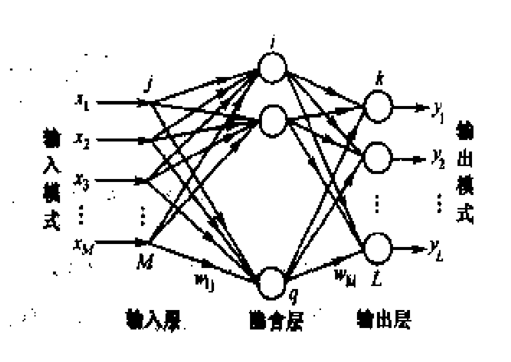


图 1-17　BP 网络的结构

BP 网络结构与多层感知机结构图相比，二者是类似的，但差异也是显著的。首先，多层感知机结构中只有一层权值可调，其他各层权值是固定的、不可学习的；BP 网络的每一层连接权值都可通过学习来调节。其次，感知机结构中的处理单元为二进制的 0 或 1；而 BP 网络的基本处理单元（输入层除外）为非线性的输入输出关系，一般选用 S 型函数。

2. BP 网络的前馈计算

在训练网络的学习阶段，设有 N 个训练样本，先假定用其中的某一个样本 p 的输入/输出模式对 $\{X_p\}$ 和 $\{T_p\}$ 对网络进行训练，隐含层的第 i 个神经元在样本 p 作用下的输入为

$$\mathrm{net}_i^p=\sum_{j=1}^{M}w_{ij}o_j^p-\theta_i=\sum_{j=1}^{M}w_{ij}x_j^p-\theta_i \quad (i=1,\ 2,\ \cdots,\ q) \tag{1-11}$$

式中，x_j^p 和 o_j^p 分别为输入节点 j 在样本 p 作用时的输入和输出，对输入节点而言两者相当；w_{ij} 为输入层神经元 j 与隐含层神经元 i 之间的连接权值；θ_i 为隐含层神经元 i 的阈值；M 为输入层的节点数，即输入的个数。

隐含层第 i 个神经元的输出为

$$o_i^p=g(\mathrm{net}_i^p) \quad (i=1,\ 2,\ \cdots,\ q) \tag{1-12}$$

式中，$g(\bullet)$ 为激活函数。

对于 Sigmoid 型激活函数

$$g(x)=\frac{1}{1+\exp[-(x+\theta_1)/\theta_0]} \tag{1-13}$$

式中，参数 θ_1 表示偏值，正的 θ_1 使激活函数水平向左移动；θ_0 的作用是调节 Sigmoid 函数形状的，较小的 θ_0 使 Sigmoid 函数逼近一个阶跃限幅函数，而较大的 θ_0 将使 Sigmoid 函数变得较为平坦，如图 1-18 所示。

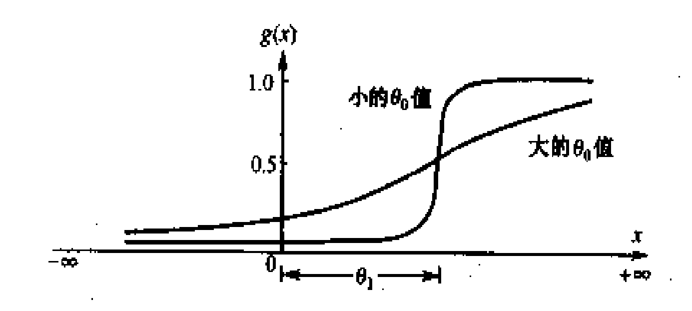


图 1-18　具有偏值的和形状调节的 Sigmoid 函数

隐含层激活函数 $g(\mathrm{net}_i^p)$ 的微分函数为

$$g'(\mathrm{net}_i^p)=g(\mathrm{net}_i^p)[1-g(\mathrm{net}_i^p)]=o_i^p(1-o_i^p) \quad (i=1,\ 2,\ \cdots,\ q) \tag{1-14}$$

隐含层第 i 个神经元的输出 o_i^p 将通过权系数向前传播到输出层第 k 个神经元并作为它的输入之一，而输出层第 k 个神经元的总输入为

$$\mathrm{net}_k^p=\sum_{i=1}^{q}w_{ki}o_i^p-\theta_k \quad (k=1, 2,\ \cdots, L) \tag{1-15}$$

式中，w_{ki} 为隐含层神经元 i 与输出层神经元 k 之间的连接权值；θ_k 为输出层神经元 k 的阈值；q 为隐含层的节点数。

输出层的第 k 个神经元的实际输出为

$$o_k^p=g(\mathrm{net}_k^p) \quad (k=1, 2,\ \cdots, L) \tag{1-16}$$

输出层激活函数 $g(\mathrm{net}_k^p)$ 的微分函数为

$$g'(\mathrm{net}_k^p)=g(\mathrm{net}_k^p)[1-g(\mathrm{net}_k^p)]=o_k^p(1-o_k^p)\quad (k=1,2,\cdots,L) \tag{1-17}$$

若其输出与给定模式对的期望输出 t_k^p 不一致，则将其误差信号从输出端反向传播回来，并在传播过程中对加权系数不断修正，直到在输出层神经元上得到所需要的期望输出值 t_k^p 为止。对样本 p 完成网络权系数的调整后，再送入另一样本模式对进行类似学习，直到完成 N 个样本的训练学习为止。

3. BP 网络权系数的调整规则

对于每一样本 p 的输入模式对的二次型误差函数为

$$J_p=\frac{1}{2}\sum_{k=1}^{L}(t_k^p-o_k^p)^2 \tag{1-18}$$

则系统对所有 N 个训练样本的总误差函数为

$$J=\sum_{p=1}^{N}J_p=\frac{1}{2}\sum_{p=1}^{N}\sum_{k=1}^{L}(t_k^p-o_k^p)^2 \tag{1-19}$$

式中，N 为模式样本对数；L 为网络输出节点数。

一般来说，基于 J_p 还是基于 J 来完成加权系数空间的梯度搜索会获得不同的结果。在 Rumelhart 等人的学习加权的规则中，学习过程按使误差函数 J_p 减小最快的方向调整加权系数直到获得满意的加权系数集为止。这里加权系数的修正是顺序操作的，网络对各模式对一个一个地顺序输入并不断进行学习，类似于生物神经网络的处理过程，但不是真正的梯度搜索过程。系统最小误差的真实梯度搜索方法是基于式（1-19）的最小化方法。

1）输出层权系数的调整

权系数应按 J_p 函数梯度变化的反方向调整，使网络逐渐收敛。根据梯度法，可得输出层每个神经元权系数的修整公式为

$$\Delta w_{ki}=-\eta\frac{\partial J_p}{\partial w_{ki}}=-\eta\frac{\partial J_p}{\partial \mathrm{net}_k^p}\cdot\frac{\partial \mathrm{net}_k^p}{\partial w_{ki}}=-\eta\frac{\partial J_p}{\partial \mathrm{net}_k^p}\cdot\frac{\partial}{\partial w_{ki}}\left(\sum_{i=1}^{q}w_{ki}o_i^p-\theta_k\right)=-\eta\frac{\partial J_p}{\partial \mathrm{net}_k^p}\cdot o_i^p \tag{1-20}$$

式中，η 为学习速率，$\eta>0$。

定义　$$\delta_k^p=-\frac{\partial J_p}{\partial \mathrm{net}_k^p}=-\frac{\partial J_p}{\partial o_k^p}\cdot\frac{\partial o_k^p}{\partial \mathrm{net}_k^p}=(t_k^p-o_k^p)\cdot g'(\mathrm{net}_k^p)=(t_k^p-o_k^p)o_k^p(1-o_k^p)$$

因此输出层的任意神经元 k 的加权系数修整公式为

$$\Delta w_{ki}=\eta\delta_k^p o_i^p=\eta o_k^p(1-o_k^p)(t_k^p-o_k^p)o_i^p \tag{1-21}$$

式中，o_k^p 为输出节点 k 在样本 p 作用时的输出；o_i^p 为隐含节点 i 在样本 p 作用时的输出；t_k^p 为在样本 p 输入输出对作用时输出节点 k 的目标值。

2）隐含层权系数的调整

根据梯度法，可得隐含层每个神经元权系数的修整公式为

$$\Delta w_{ij}=-\eta\frac{\partial J_p}{\partial w_{ij}}=-\eta\frac{\partial J_p}{\partial \mathrm{net}_i^p}\cdot\frac{\partial \mathrm{net}_i^p}{\partial w_{ij}}=-\eta\frac{\partial J_p}{\partial \mathrm{net}_i^p}\cdot\frac{\partial}{\partial w_{ij}}(\sum_{j=1}^{M}w_{ij}o_j^p-\theta_i)$$

$$=-\eta\frac{\partial J_p}{\partial \mathrm{net}_i^p}\cdot o_j^p=\eta\delta_i^p o_j^p$$

式中，η 为学习速率，$\eta>0$。

$$\delta_i^p=-\frac{\partial J_p}{\partial \mathrm{net}_i^p}=-\frac{\partial J_p}{\partial o_i^p}\cdot\frac{\partial o_i^p}{\partial \mathrm{net}_i^p}=-\frac{\partial J_p}{\partial o_i^p}\cdot g'(\mathrm{net}_i^p)=-\frac{\partial J_p}{\partial o_i^p}\cdot o_i^p(1-o_i^p)$$

由于隐含层一个单元输出的改变会影响与该单元相连接的所有输出单元的输入，即

$$-\frac{\partial J_p}{\partial o_i^p}=-\sum_{k=1}^{L}\frac{\partial J_p}{\partial \mathrm{net}_k^p}\cdot\frac{\partial \mathrm{net}_k^p}{\partial o_i^p}=-\sum_{k=1}^{L}\frac{\partial J_p}{\partial \mathrm{net}_k^p}\cdot\frac{\partial}{\partial o_i^p}(\sum_{i=1}^{q}w_{ki}o_i^p-\theta_k)$$

$$=\sum_{k=1}^{L}\left(-\frac{\partial J_p}{\partial \mathrm{net}_k^p}\right)\cdot w_{ki}=\sum_{k=1}^{L}\delta_k^p\cdot w_{ki}$$

因此，隐含层的任意神经元 i 的加权系数修正公式为

$$\Delta w_{ij}=\eta\delta_i^p o_j^p=\eta o_i^p(1-o_i^p)(\sum_{k=1}^{L}\delta_k^p\cdot w_{ki})o_j^p \tag{1-22}$$

式中，o_i^p 为隐含节点 i 在样本 p 作用时的输出；o_j^p 为输入节点 j 在样本 p 作用时的输出，即输入节点 j 的输入（因对输入节点两者相当）。

输出层的任意神经元 k 在样本 p 作用时的加权系数改进公式为

$$w_{ki}(k+1)=w_{ki}(k)+\eta\delta_k^p o_i^p \tag{1-23}$$

隐含层的任意神经元 i 在样本 p 作用时的加权系数改进公式为

$$w_{ij}(k+1)=w_{ij}(k)+\eta\delta_i^p o_j^p \tag{1-24}$$

由式（1-23）和式（1-24）可知，对于给定的某一个样本 p，可根据误差要求调整网络的加权系数使其满足要求；对于给定的另一个样本，再根据误差要求调整网络的加权系数使其满足要求，直到所有样本作用下的误差都满足要求为止。这种计算过程称在线学习过程。

如果学习过程按使误差函数 J 减小最快的方向调整加权系数，采用类似的推导过程可得输出层和隐含层的任意神经元 k 和 i 在所有样本作用时的加权系数增量公式为

$$w_{ki}(k+1)=w_{ki}(k)+\eta\sum_{p=1}^{N}\delta_k^p o_i^p \tag{1-25}$$

$$w_{ij}(k+1)=w_{ij}(k)+\eta\sum_{p=1}^{N}\delta_i^p o_j^p \tag{1-26}$$

式中，δ_k^p 和 δ_i^p 的计算方法同上，即

$$\delta_k^p=o_k^p(1-o_k^p)(t_k^p-o_k^p) \tag{1-27}$$

$$\delta_i^p = o_i^p(1-o_i^p)(\sum_{k=1}^{L}\delta_k^p \cdot w_{ki}) \tag{1-28}$$

根据式（1-25）和式（1-26）所得权的修正是在所有样本输入后，计算完总的误差后进行的，这种修正称为批处理学习或称为离线学习。批处理修正可保证其总误差 J 向减少的方向变化，在样本多的时候，它比在线学习的收敛速度快。

因此，BP 网络的学习可采用两种方式，即在线学习和离线学习。在线学习是对训练集内每个模式对逐一更新网络权值的一种学习方式，其特点是学习过程中需要较少的存储单元，但有时会增加网络的整体输出误差，以上推导即为在线学习过程。因此，使用在线学习时一般应使学习因子足够小，以保证用训练集内每个模式训练一次后，权值的总体变化充分接近于最快速下降。所谓离线学习也称为批处理学习，是指用训练集内所有模式依次训练网络，累加各权值修正量并统一修正网络权值的一种学习方式，它能使权值变化沿最快速下降方向进行。其缺点是学习过程中需要较多的存储单元，好处是学习速度较快。在具体实际应用中，当训练模式很多时，可以将整个训练模式分成若干组，采用分组批处理学习方式。

4. BP 网络学习算法的计算步骤

（1）初始化：置所有的加权系数为最小的随机数；

（2）提供训练集：给出顺序赋值的输入向量 $x_1, x_2, \cdots, x_N$ 和期望的输出向量 $t_1, t_2, \cdots, t_N$；

（3）计算实际输出：按式（1-11）～式（1-17）计算隐含层、输出层各神经元的输出；

（4）按式（1-18）或式（1-19）计算期望值与实际输出的误差；

（5）按式（1-23）或式（1-25）调整输出层的加权系数 w_{ki}；

（6）按式（1-24）或式（1-26）调整隐含层的加权系数 w_{ij}；

（7）返回计算（3）步，直到误差满足要求为止。

5. 使用 BP 算法时应注意的几个问题

（1）学习速率 η 的选择非常重要。在学习初始阶段，η 选得大些可使学习速度加快，但当临近最佳点时，η 必须相当小；否则，加权系数将产生反复振荡而不能收敛。采用变学习速率方案，令学习速率 η 随着学习的进展而逐步减小，可收到较好的效果。引入惯性系数 α 的办法，也可使收敛速度加快，α 的取值可选在 0.9 左右。

（2）采用 S 型激活函数式（1-13）时，由于输出层各神经元的理想输出值只能接近于 1 或 0，而不能达到 1 或 0。因此在设置各训练样本的期望输出分量 t_k^p 时，不能设置为 1 或 0，以设置为 0.9 或 0.1 较为适宜。

（3）由式（1-13）及图 1-18 可知，S 型非线性激活函数 $g(x)$ 随着 $|x|$ 的增大梯度下降，即 $|g'(x)|$ 减小并趋于 0，不利于权值的调整，因此希望 $|x|$ 工作在较小的区域，故应考虑网络的输入。若实际问题给予网络的输入量较大，需做归一化处理，网络的输出也要进行相应的处理。对于具体问题，需经调试而定，且需经验知识的积累。

（4）学习开始时如何设置加权系数 w_{ij} 和 w_{ki} 的初值非常重要，如将所有初值设置为相等值，则由式（1-22）可知，所有隐含层加权系数的调整量相同，从而使这些加权系数总相等，因此各加权系数的初值以设置为随机数为宜。

（5）BP 网络的另一个问题是学习过程中系统可能陷入某些局部最小值，或某些静态点，或在这些点之间振荡。在这种情况下，不管进行多少次迭代，系统都存在很大误差。因此，在学习过程中，应尽量避免落入某些局部最小值点上，引入惯性项有可能使网络避免落入某一局部最小值。

6. BP 网络学习算法的改进

BP 网络总括起来，具有以下主要优点：

（1）只要有足够多的隐含层和隐节点，BP 网络可以逼近任意的非线性映射关系；

（2）BP 网络的学习算法属于全局逼近的方法，因而它具有较好的泛化能力。

它的主要缺点是：

（1）收敛速度慢；

（2）局部极值；

（3）难以确定隐含层和隐节点的个数。

从原理上讲，只要有足够多的隐含层和隐节点，即可实现复杂的映射关系，但是如何根据特定的问题来具体确定网络的结构尚无很好的方法，仍需要凭借经验和试凑。

BP 网络能够实现输入输出的非线性映射关系，但它并不依赖于模型。其输入与输出之间的关联信息分布地存储于连接权中。由于连接权的个数很多，个别神经元的损坏只对输入输出关系有较小的影响，因此 BP 网络显示了较好的容错性。

BP 网络由于其很好的逼近非线性映射的能力，因而它可应用于信息处理、图像识别、模型辨识、系统控制等多个方面。对于控制方面的应用，其很好的逼近特性和泛化能力是一个优点。而收敛速度慢却是一个很大的缺点，这一点难以满足具有适应功能的实时控制的要求，它影响了该网络在许多方面的实际应用。为此，许多人对 BP 网络的学习算法进行了广泛的研究，提出了许多改进的算法，下面介绍典型的几种。

1）引入惯性项

有时为了使收敛速度快些，可在加权系数修正公式中增加一个惯性项，使加权系数变化更平稳些。

输出层的任意神经元 k 在样本 p 作用时的加权系数改进公式为

$$w_{ki}(k+1)=w_{ki}(k)+\eta\delta_k^p o_i^p+\alpha[w_{ki}(k)-w_{ki}(k-1)] \tag{1-29}$$

隐含层的任意神经元 i 在样本 p 作用时的加权系数改进公式为

$$w_{ij}(k+1)=w_{ij}(k)+\eta\delta_i^p o_j^p+\alpha[w_{ij}(k)-w_{ij}(k-1)] \tag{1-30}$$

式中，α为惯性系数，$0<\alpha<1$。

2）引入动量项

上述标准 BP 算法实质上是一种简单的最速下降静态寻优算法，在修正 $w(k)$时，只是按 k 时刻的负梯度方式进行修正，而没有考虑以前积累的经验，即以前时刻的梯度方向，从而常常使学习过程发生振荡，收敛缓慢。为此，有人提出了如下的改进算法，即

$$w(k+1)=w(k)+\eta[(1-\alpha)D(k)+\alpha D(k-1)] \tag{1-31}$$

式中，$w(k)$既可表示单个的连接权系数，也可表示连接权向量（其元素为连接权系数）；$D(k)=-\partial J/\partial w(k)$为 k 时刻的负梯度；$D(k-1)$是 $k-1$ 时刻的负梯度；η 为学习速率，$\eta>0$；α为动量项因子，$0\leqslant\alpha<1$。

该方法所加入的动量项实质上相当于阻尼项，它减小了学习过程的振荡趋势，改善了收敛性，这是目前应用比较广泛的一种改进算法。

3）变尺度法

标准的 BP 学习算法所采用的是一阶梯度法，因而收敛较慢。若采用二阶梯度法，则可以大大改善收敛性。二阶梯度法的算法为

$$w(k+1)=w(k)+\eta[\nabla^2 J(k)]^{-1}\nabla J(k)$$

式中，$\nabla J(k)=\dfrac{\partial J}{\partial w(k)},\nabla^2 J(k)=\dfrac{\partial^2 J}{\partial w^2(k)},0<\eta\leqslant 1$。

虽然二阶梯度法具有比较好的收敛性，但是它需要计算 J 对 $w(k)$的二阶导数，这个计算量是很大的。一般不直接采用二阶梯度法，而常常采用变尺度法或共轭梯度法，它们具有二阶梯度法收敛较快的优点，而又无需直接计算二阶梯度。下面具体给出变尺度法的算法，即

$$w(k+1)=w(k)+\eta H(k)D(k)$$

$$H(k)=H(k-1)-\frac{\Delta w(k)\Delta w^{\mathrm{T}}(k)}{\Delta w^{\mathrm{T}}(k)\Delta D(k)}-\frac{H(k-1)\Delta D(k)\Delta D^{\mathrm{T}}(k)H(k-1)}{\Delta D^{\mathrm{T}}(k)H(k-1)\Delta D(k)}$$

$$\Delta w(k)=w(k)-w(k-1)$$

$$\Delta D(k)=D(k)-D(k-1)$$

4）变步长法

一阶梯度法寻优收敛较慢的一个重要原因是η（学习速率）不好选择。η 选得太小，收敛太慢，选得太大，则有可能修正过头，导致振荡甚至发散。为了解决这个问题，提出了如下的变步长算法，即

$$w(k+1)=w(k)+\eta(k)D(k)$$

$$\eta(k)=2^{\lambda}\eta(k-1)$$

$$\lambda=\mathrm{sgn}[D(k)D(k-1)]$$

以上公式说明，当连续两次迭代其梯度方向相同时，表明下降太慢，这时可使步长加倍；当连续两次迭代其梯度方向相反时，表明下降过头，这时可使步长减半。需要引入动量项时，上述算法可修正改为

$$w(k+1)=w(k)+\eta(k)[(1-\eta)D(k)+\eta D(k-1)]$$
$$\eta(k)=2^{\lambda}\eta(k-1)$$
$$\lambda=\text{sgn}[D(k)D(k-1)]$$

在使用该算法时，由于步长在迭代过程中自适应进行调整，因此对于不同的连接权系数实际采用了不同的学习速率，也就是说误差代价函数 J 在超曲面上在不同的方向按照各自比较合理的步长向极小点逼近。

例 1-3 一个具有两个输入单元，两个隐含单元和一个输出单元的两层 BP 神经网络，如图 1-19 所示。设学习样本为 N=4，写出 BP 网络学习算法批处理（离线学习）的计算步骤。假设样本集的输入为 $X=\{x^1,x^2,x^3,x^4\}$，目标为 $T=\{t^1,t^2,t^3,t^4\}$。$x^p=[x_1^p,x_2^p]$（p=1, 2, 3, 4）。

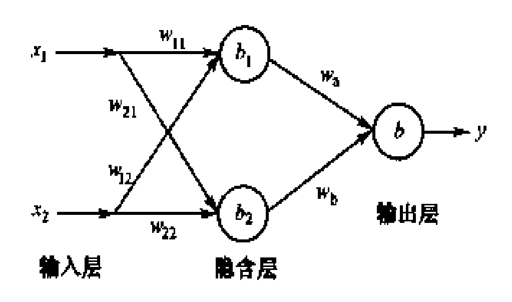


图 1-19 简单 BP 网络

解：（1）置所有的加权系数及偏值的初始值为最小的随机数，即

隐含层的加权系数及偏值的初始值为：$w_{11}(0),w_{12}(0),w_{21}(0),w_{22}(0),b_1(0),b_2(0)$

输出层的加权系数及偏值的初始值为：$w_a(0),w_b(0),b(0)$

（2）根据式（1-12）和式（1-16）分别计算样本集中所有样本的隐含层和输出层各节点的输出值，即

隐含层第 1 个神经元的输出为：$o_1^p=g(w_{11}\cdot x_1^p+w_{12}\cdot x_2^p+b_1)$ （p=1,2,3,4）

隐含层第 2 个神经元的输出为：$o_2^p=g(w_{21}\cdot x_1^p+w_{22}\cdot x_2^p+b_2)$ （p=1,2,3,4）

输出层神经元的输出为：$y^p=g(w_a\cdot o_1^p+w_b\cdot o_2^p+b)$ （p=1,2,3,4）

（3）根据式（1-21）和式（1-22）及目标值分别计算在所有样本作用下的各层误差，即

输出层的误差为：$\delta^p=y^p(1-y^p)(t^p-y^p)$ （p=1,2,3,4）

隐含层第 1 个神经元的误差为：$\delta_1^p=\delta^p\cdot w_a\cdot o_1^p(1-o_1^p)$ （p=1,2,3,4）

隐含层第 2 个神经元的误差为：$\delta_2^p=\delta^p\cdot w_b\cdot o_2^p(1-o_2^p)$ （p=1,2,3,4）

（4）根据式（1-25）和式（1-26）调整各层的加权系数及偏值，即

输出层的加权系数及阈值修正公式为

$$w_a(k+1)=w_a(k)+\eta\sum_{p=1}^{4}\delta^p o_1^p\text{，}\quad w_b(k+1)=w_b(k)+\eta\sum_{p=1}^{4}\delta^p o_2^p$$

$$b(k+1)=b(k)+\eta\sum_{p=1}^{4}\delta^{p}$$

隐含层的加权系数及阈值修正公式为

$$w_{11}(k+1)=w_{11}(k)+\eta\sum_{p=1}^{4}\delta_1^p x_1^p\text{，}\quad w_{12}(k+1)=w_{12}(k)+\eta\sum_{p=1}^{4}\delta_1^p x_2^p$$

$$w_{21}(k+1)=w_{21}(k)+\eta\sum_{p=1}^{4}\delta_2^p x_1^p\text{，}\quad w_{22}(k+1)=w_{22}(k)+\eta\sum_{p=1}^{4}\delta_2^p x_2^p$$

$$b_1(k+1)=b_1(k)+\eta\sum_{p=1}^{4}\delta_1^p\text{，}\quad b_2(k+1)=b_2(k)+\eta\sum_{p=1}^{4}\delta_2^p$$

（5）计算输出误差

$$J=\frac{1}{2}[(t^1-y^1)^2+(t^2-y^2)^2+(t^3-y^3)^2+(t^4-y^4)^2]$$

存在一个$\varepsilon>0$，使 $J<\varepsilon$；否则，返回（2）重新计算，直到误差满足要求为止。

例 1-4　利用一个三输入两输出的两层 BP 神经网络训练一个输入为[1 1;-1 -1;1 1]，期望的输出均为[1,1]的神经网络系统。激活函数取

$$g(x)=\frac{2}{1+\exp(-x)}-1$$

解：根据 BP 网络学习算法的计算步骤，用 MATLAB 语言编写的在线学习程序如下：

```
% BP 网络的第一阶段学习期（训练加权系数 wki,wij）
%初始化
lr=0.05;err_goal=0.001;              %lr 为学习速率;err_goal 为期望误差最小值
max_epoch=10000;a=0.9;               %max_epoch 为训练的最大次数；a 为惯性系数
Oi=0;Ok=0;                           %置隐含层和输出层各神经元输出初值为零
%提供两组训练集和目标值（3 输入，2 输出）
X=[1 1;-1 -1;1 1];T=[1 1;1 1];
%初始化 wki,wij（M 为输入节点 j 的数量；q 为隐含层节点 i 的数量；L 为输出节点 k 的数量）
[M,N]=size(X);q=8;[L,N]=size(T);     %N 为训练集对数量
wij=rand(q,M);wki=rand(L,q);
wij0=zeros(size(wij));wki0=zeros(size(wki));
for epoch=1:max_epoch
%计算隐含层各神经元输出
NETi=wij*X;
for j=1:N
  for i=1:q
    Oi(i,j)=2/(1+exp(-NETi(i,j)))-1;
```

```
    end
  end
  %计算输出层各神经元输出
  NETk=wki*Oi;
  for i=1:N
    for k=1:L
      Ok(k,i)=2/(1+exp(-NETk(k,i)))-1;
    end
  end
  %计算误差函数
  E=((T-Ok)'*(T-Ok))/2;
  if (E<err_goal) break;end
  %调整输出层加权系数
  deltak=Ok.*(1-Ok).*(T-Ok);
  w=wki;
  wki=wki+lr*deltak*Oi'+a*(wki-wki0);
  wki0=w;
  %调整隐含层加权系数
  deltai=Oi.*(1-Oi).*(deltak'*wki)';
  w=wij;
  wij=wij+lr*deltai*X'+a*(wij-wij0);
  wij0=w;
  end
  epoch                                  %显示计算次数
  %BP 网络的第二阶段工作期（根据训练好的 wki,wij 和给定的输入计算输出）
  X1=X;                                  %给定输入
  %计算隐含层各神经元输出
  NETi=wij*X1;
  for j=1:N
    for i=1:q
      Oi(i,j)=2/(1+exp(-NETi(i,j)))-1;
    end
  end
  %计算输出层各神经元输出
  NETk=wki*Oi;
  for i=1:N
```

```
    for k=1:L
      Ok(k,i)=2/(1+exp(-NETk(k,i)))-1;
    end
  end
  Ok                                    %显示网络输出层的输出
```

其结果显示为

```
  epoch =
       3
  Ok =
     0.9826    0.9826
     0.9948    0.9948
```

1.2.5 径向基神经网络

1985 年，Powell 提出了多变量插值的径向基函数（Radial Basis Function，RBF）方法。1988 年，Broomhead 和 Lowe 首先将 RBF 应用于神经网络设计，从而构成了 RBF 神经网络。它是一种局部逼近的神经网络。众所周知，BP 网络用于函数逼近时，权值的调节采用的是负梯度下降法，这种调节权值的方法有它的局限性，即存在着收敛速度慢和局部极小等缺点。而 RBF 神经网络无论在逼近能力、分类能力和学习速度等方面均优于 BP 网络。径向基函数网络比 BP 网络需要更多的神经元，但是它能够按时间片来训练网络。径向基网络是一种局部逼近网络，已证明它能以任意精度逼近任一连续函数。当有很多的训练向量时，这种网络很有效果。

RBF 网络的结构与多层前向网络类似，但它是具有单隐含层的一种两层前向网络。输入层由信号源节点组成。隐含层的单元数视所描述问题的需要而定。输出层对输入的作用做出响应。从输入空间到隐含层空间的变换是非线性的，而从隐含层空间到输出层空间的变换是线性的。隐单元的变换函数是 RBF，它是一种局部分布的对中心点径向对称衰减的非负非线性函数。

构成 RBF 网络的基本思想是：用 RBF 作为隐单元的“基”构成隐含层空间，这样就可将输入矢量直接（即不通过权连接）映射到隐空间。当 RBF 的中心点确定以后，这种映射关系也就确定了。而隐含层空间到输出空间的映射是线性的，即网络的输出是隐单元输出的线性加权和。此处的权即为网络可调参数。由此可见，从总体上看，网络由输入到输出的映射是非线性的，而网络输出对可调参数而言却又是线性的。这样，网络的权就可由线性方程组直接解出或用 LMS 方法计算，从而大大加快学习速度并避免局部极小问题。

1．径向基函数网络模型

RBF 网络由两层组成，其结构如图 1-20 所示。输入层节点只是传递输入信号到隐含

层，隐含层节点（也称 RBF 节点）由像高斯核函数那样的辐射状作用函数构成，而输出层节点通常是简单的线性函数。

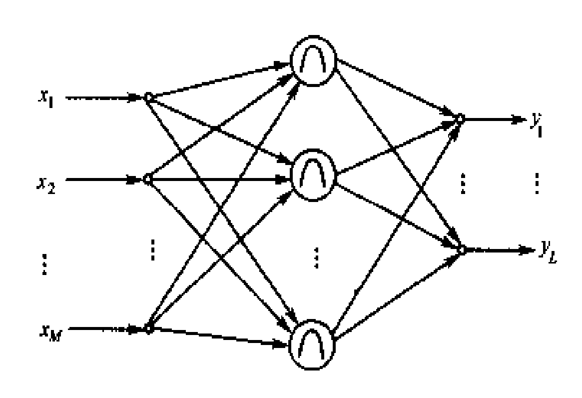


图 1-20 RBF 网络的结构

隐含层节点中的作用函数（核函数）对输入信号将在局部产生响应，也就是说，当输入信号靠近该函数的中央范围时，隐含层节点将产生较大的输出。由此可看出这种网络具有局部逼近能力，故径向基函数网络也称为局部感知场网络。

2. 网络输出

设网络输入 $\boldsymbol{x}$ 为 M 维向量，输出 $\boldsymbol{y}$ 为 L 维向量，输入输出样本对长度为 N。

RBF 网络的输入层到隐含层实现 $\boldsymbol{x}\to u_i(\boldsymbol{x})$的非线性映射，径向基网络隐含层节点的作用函数一般取下列几种形式，即

$$u_i(\boldsymbol{x})=\exp[-(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}/\delta_i^2)]$$

$$u_i(\boldsymbol{x})=\frac{1}{(\delta_i^2+\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x})^{\alpha}},\alpha>0$$

$$u_i(\boldsymbol{x})=(\delta_i^2+\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x})^{\beta},\alpha<\beta<1$$

上面这些函数都是径向对称的，虽然有各种各样的激活函数，但最常用的是高斯激活函数，如 RBF 网络隐含层第 i 个节点的输出可由下式表示，即

$$u_i=\exp\left[-\frac{(\boldsymbol{x}-\boldsymbol{c}_i)^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{c}_i)}{2\sigma_i^2}\right]\qquad(i=1,2,\cdots,q)\tag{1-32}$$

式中，u_i 是第 i 个隐节点的输出，σ_i 是第 i 个隐节点的标准化常数，q 是隐含层节点数，$\boldsymbol{x}=(x_1, x_2,\cdots, x_M)^{\mathrm{T}}$ 是输入样本；$\boldsymbol{c}_i$ 是第 i 个隐节点高斯函数的中心向量，此向量是一个与输入样本 $\boldsymbol{x}$ 的维数相同的列向量，即 $\boldsymbol{c}_i=(c_{i1}, c_{i2},\cdots, c_{iM})^{\mathrm{T}}$。由上式可知，节点的输出范围在 0 和 1 之间，且输入样本愈靠近节点的中心，输出值愈大。当 $\boldsymbol{x}=\boldsymbol{c}_i$ 时，$u_i=1$。

采用高斯基函数，具备如下优点：

（1）表示形式简单，即使对于多变量输入也不增加太多的复杂性；

（2）径向对称；

（3）光滑性好，任意阶导数存在；

（4）由于该基函数表示简单且解析性好，因而便于进行理论分析。

考虑到提高网络精度和减少隐含层节点数，也可以将网络激活函数改成多变量正态密度函数，即

$$u_i = \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{c}_i)K(\boldsymbol{x}-\boldsymbol{c}_i)^{\mathrm{T}}\right] \tag{1-33}$$

式中，$K = E[(\boldsymbol{x}-\boldsymbol{c}_i)^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{c}_i)^{-1}]$ 是输入协方差阵的逆。这时上式已不再是径向对称。

RBF 网络的隐含层到输出层实现 $u_i(\boldsymbol{x}) \to y_k$ 的线性映射，即

$$y_k = \sum_{i=1}^{q} w_{ki} u_i - \theta_k \qquad (k=1,\ 2,\ \cdots,\ L) \tag{1-34}$$

式中，u_i 是隐含层第 i 个节点的输出；y_k 是输出层第 k 个节点的输出；w_{ki} 是隐含层到输出层的加权系数；θ_k 是输出层的阈值；q 是隐含层节点数。

3. RBF 网络的学习过程

设有 N 个训练样本，则系统对所有 N 个训练样本的总误差函数为

$$J = \sum_{p=1}^{N} J_p = \frac{1}{2}\sum_{p=1}^{N}\sum_{k=1}^{L}(t_k^p - y_k^p)^2 = \frac{1}{2}\sum_{p=1}^{N}\sum_{k=1}^{L} e_k^2 \tag{1-35}$$

式中，N 为模式样本对数；L 为网络输出节点数；t_k^p 表示在样本 p 作用下的第 k 个神经元的期望输出；y_k^p 表示在样本 p 作用下的第 k 个神经元的实际输出。

RBF 网络的学习过程分为两个阶段。第一阶段是无教师学习，是根据所有的输入样本决定隐含层各节点的高斯核函数的中心向量 c_i 和标准化常数 σ_i。第二阶段是有教师学习。在决定好隐含层的参数后，根据样本，利用最小二乘原则，求出隐含层和输出层的权值 w_{ki}。有时在完成第二阶段的学习后，再根据样本信号，同时校正隐含层和输出层的参数，以进一步提高网络的精度。下面具体介绍一下这两个阶段。

1）无教师学习阶段

无教师学习也称为非监督学习，是对所有样本的输入进行聚类，求得各隐含层节点的 RBF 的中心向量 c_i。这里介绍 k-均值聚类算法调整中心向量，此算法将训练样本集中的输入向量分为若干族，在每个数据族内找出一个径向基函数中心向量，使得该族内各样本向量距该族中心的距离最小。算法步骤如下：

（1）给定各隐节点的初始中心向量 $c_i(0)$ 和判定停止计算的 ε；

（2）计算距离（欧氏距离）并求出最小距离的节点；

$$\begin{cases} d_i(k) = \|\boldsymbol{x}(k) - \boldsymbol{c}_i(k-1)\|, 1 \leqslant i \leqslant q \\ d_{\min}(k) = \min d_i(k) = d_r(k) \end{cases} \tag{1-36}$$

式中，k 为样本序号，r 为中心向量 $c_i(k-1)$ 与输入样本 $\boldsymbol{x}(k)$ 距离最近的隐节点序号；

（3）调整中心

$$\begin{cases} c_i(k)=c_i(k-1), 1\leqslant i\leqslant q, i\leqslant r \\ c_r(k)=c_r(k-1)+\beta(k)[x(t)-c_r(k-1)] \end{cases} \tag{1-37}$$

式中，$\beta(k)$是学习速率。$\beta(k)=\beta(k-1)/(1+\mathrm{int}(k/q))^{1/2}$, int(•)表示对(•)进行取整运算。可见，每经过 q 个样本之后，调小一次学习速率，逐渐减至零；

（4）判定聚类质量

对于全部样本 $k(k=1,2,\cdots,N)$反复进行以上（2），（3）步，直至满足以上条件，则聚类结束。

若

$$J_e=\sum_{i=1}^{q}\|x(k)-c_i(k)\|^2\leqslant\varepsilon \tag{1-38}$$

2）有教师学习阶段

有教师学习也称为有监督学习。当 c_i 确定以后，训练由隐含层至输出层之间的权值，由上可知，它是一个线性方程组，则求权值就成为线性优化问题。因此，问题有惟一确定的解，不存在 BP 网络中所遇到的局部极小值问题，肯定能获得全局最小点。类似于线性网络，RBF 网络的隐含层至输出层之间的连接权值 $w_{ki}(k=1,2,\cdots,L;i=1,2,\cdots,q)$学习算法为

$$w_{ki}(k+1)=w_{ki}(k)+\eta(t_k-y_k)u_i(x)/u^{\mathrm{T}}u \tag{1-39}$$

式中，$u=[u_1(x)u_2(x)\cdots u_q(x)]^{\mathrm{T}}$，$u_i(x)$为高斯函数；$\eta$ 为学习速率。可以证明，当$0<\eta<1$时，可保证该迭代学习算法的收敛性，而实际上通常只取$0<\eta<1$；t_k和 y_k 分别表示第 k 个输出分量的期望值和实际值。由于向量 u 中只有少量几个元素为 1，其余均为 0，因此在一次数据训练中只有少量的连接权需要调整。正是由于这个特点，才使得 RBF 神经网络具有比较快的学习速度。另外，由于当 x 远离 c_i 时，$u_i(x)$非常小，因此可作为 0 对待。因此，实际上只当 $u_i(x)$大于某一数值（如 0.05）时才对相应的权值 w_{ki} 进行修改。经这样处理后，RBF 神经网络也同样具备局部逼近网络学习收敛快的优点。

4. RBF 网络有关的几个问题

（1）从理论上而言，RBF 网络和 BP 网络一样可近似任何的连续非线性函数。两者的主要不同点是在非线性映射上采用了不同的作用函数。BP 网络中的隐节点使用的是 Sigmoid 函数，其函数值在输入空间中无限大的范围内为非零值，即作用函数为全局的；而 RBF 网络中的隐节点使用的是高斯基函数，即它的作用函数则是局部的。

（2）已证明 RBF 网络具有惟一最佳逼近的特性，且无局部极小。

（3）求 RBF 网络隐节点的中心向量 c_i 和标准化常数σ_i 是一个困难的问题。

（4）径向基函数，即径向对称函数有多种。对于同一组样本，如何选择合适的径向基函数，如何确定隐节点数，以使网络学习达到要求的精度，目前还无解决办法。当前，用计算机选择、设计、再检验是一种通用的手段。

（5）RBF 网络用于非线性系统辨识与控制，虽具有惟一最佳逼近的特性以及无局部极小的优点，但隐节点的中心难求，这是该网络难以广泛应用的原因。

（6）与 BP 网络收敛速度慢的缺点相反，RBF 网络学习速度很快，适于在线实时控制。这是因为 RBF 网络把一个难题分解成了两个较易解决的问题。首先，通过若干个隐节点，用聚类方式覆盖全部样本模式；然后，修改输出层的权值，以获得最小映射误差。而这两步都比较直观。

（7）图 1-21 是径向基函数神经元传输函数 radbas( )的示意图，从图中可以注意到 RBF 网络的输入同前面介绍的神经网络的表达式有所不同。其网络输入为权值向量 $\boldsymbol{W}$ 与输入向量 $\boldsymbol{x}$ 之间的向量距离乘以阈值 $\boldsymbol{b}$，即 d = radbas (dist($\boldsymbol{W}$, $\boldsymbol{x}$)*$\boldsymbol{b}$)。由左图可知，径向基函数的输入为 0 时，输出为极大值 1。由右图可知，径向基函数的输出随权值向量 $\boldsymbol{W}$ 和输入向量 $\boldsymbol{x}$ 之间的距离减小而增大，因此可以直观地想像：当输入向量 $\boldsymbol{x}$ 同权值向量 $\boldsymbol{W}$ 完全相同时，径向基函数的输出为 1，这时，径向基函数扮演了信号检测器的角色。阈值 $\boldsymbol{b}$ 可调节 RBF 神经元的敏感程度。例如，如果一个神经元的阈值为 0.1，那么，当向量距离为 8.326 (0.8326/$\boldsymbol{b}$)时，神经元的输出为 0.5。

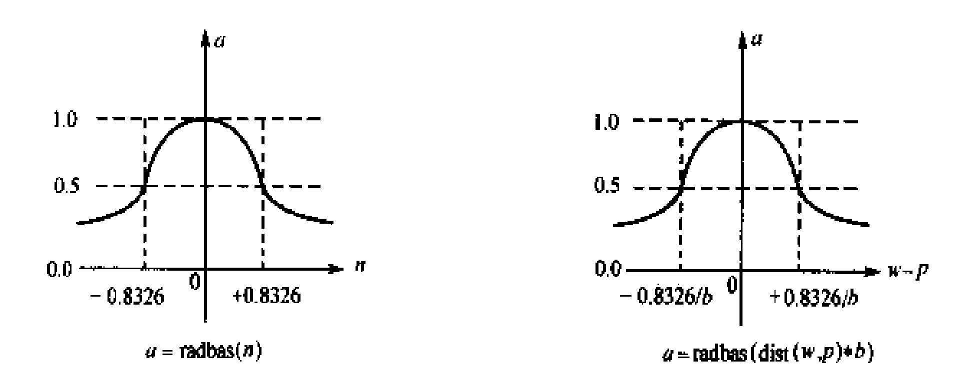


图 1-21　传输函数 radbas( )的示意图

例 1-5　对于单输入单输出，隐含层有三个节点的高斯 RBF 网络，如图 1-22 所示，已知三个隐节点的中心 c_1=−1, c_2=0, c_3=2.5；标准化参数σ_i^2=1(i=1,2,3)。三个非线性作用函数（高斯 RBF）如图 1-23 所示，隐含层至输出的权系数 $\boldsymbol{w}$=[1.1　−1　0.5]^{T}。

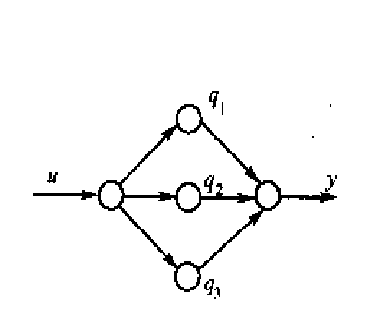


图 1-22　RBF 网络结构

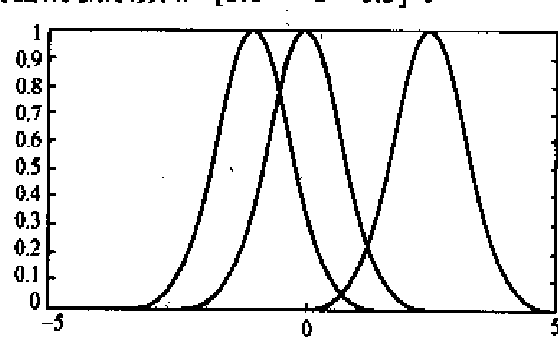


图 1-23　三个高斯 RBF

解：① 在上述条件下，设网络输入 x=-5:0.5:4.5，如图 1-24（a）所示。由式（1-32）和式（1-34）可得到网络的输出 t，如图 1-24（b）所示，即得到样本对 $x^p/t^p(p=1\sim20)$。

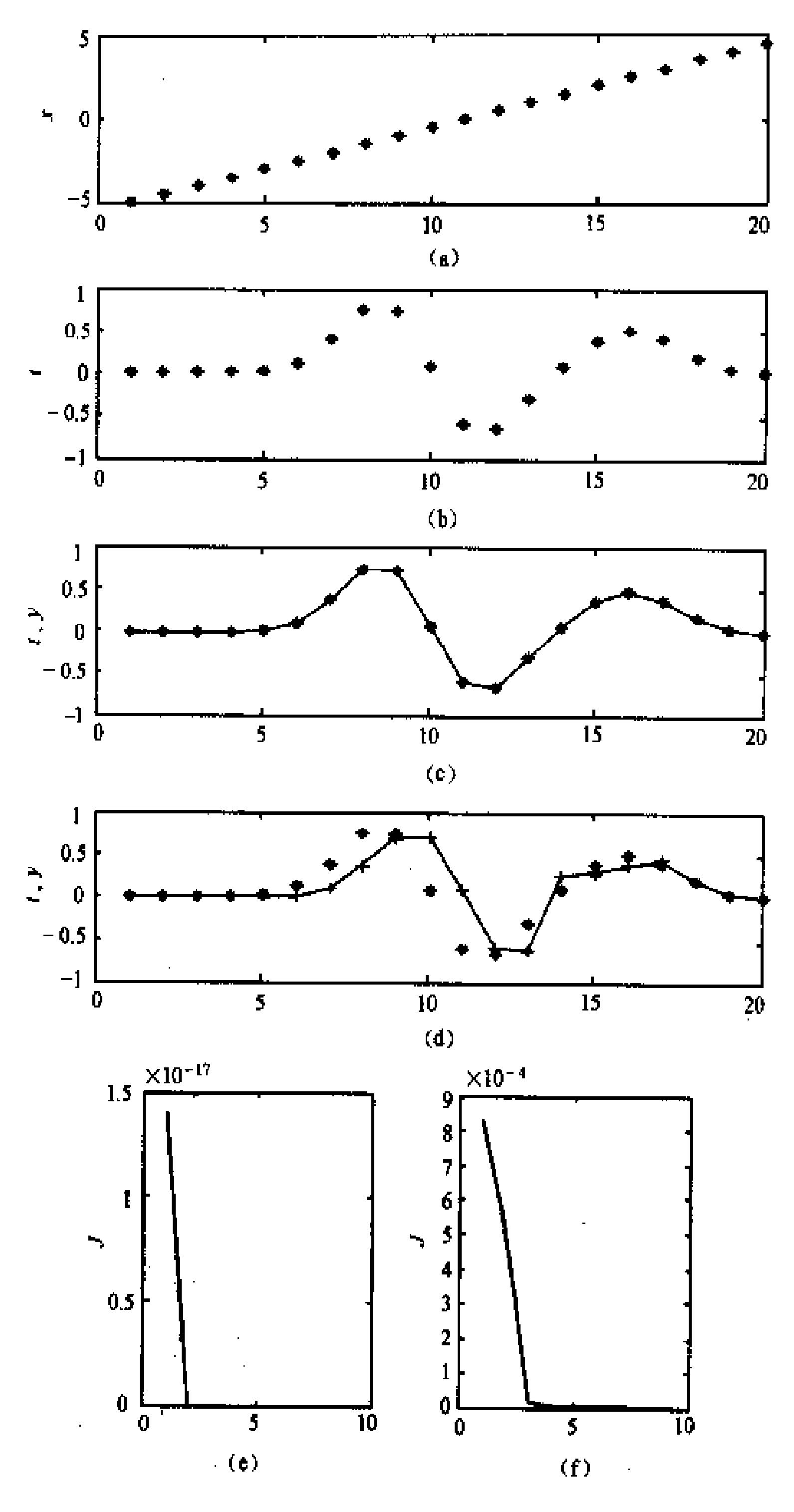


图 1-24 高斯 RBF 网络学习实例

②设隐含节点中心已知，由样本对 x^p/t^p，训练网络权系数 $w=[w_1 \quad w_2 \quad w_3]^{\mathrm{T}}$。取初始权

值 $w(0)$，采用式（1-39）训练网络，次数 $p\geq5$，网络输出与期望输出相等，即样本 $y^p=t^p$，如图 1-24（c）所示。对于目标函数式（1-35），当次数 $p\geq5$ 时，$J(p)=0$，如图 1-24（e）所示。

③ 设隐含层节点中心未知，采用 k-均值与 LMS 算法，根据式（1-36）～式（1-39）训练网络，由于隐含层节点中心很难搜索到所设值，因此网络输出与样本输出有相当的误差，$y^p\neq t^p$，如图 1-24（d）所示。当训练次数 p 增加，目标函数为非零常值，如图 1-24（f）所示。

1.2.6　竞争学习神经网络

竞争学习是一种典型无导师学习策略，是指同一神经元层次上各个神经元相互之间进行竞争，竞争胜利的神经元修改与其相联的连接权值，它模拟人类根据过去经验自动适应无法预测的环境变化。在无监督学习中，只向网络提供一些学习样本，而不提供理想的输出。网络根据输入样本进行自组织，并将其划分到相应的模式类中。因为没有教师信号，所以这类网络通常利用竞争的原则进行。学习时只需给定一个输入模式集作为训练集，网络自行组织训练模式，并将其分成不同类型。与 BP 学习相比，这种学习能力进一步拓宽了神经网络在模式识别、分类方面的应用。竞争学习网络的核心——竞争层是许多神经网络模型的重要组成部分，如自组织竞争神经网络（Self organizing NNs）、Kohonen 提出的自组织特征映射网络（SOM）、Hecht-Nielson 提出的反传网络（CPN）、Grossberg 与 Carpenter 提出的自适应共振理论（ART）网络模型等均包含竞争层。以下将分别详细讲述这些竞争网络结构的基本思想和学习规则。

1．自组织竞争神经网络（Self organizing NNs）

自组织竞争人工神经网络的形成是受生物神经系统的启发，之所以称为自组织竞争人工神经网络，是因为它一般是由输入层和竞争层构成的。自组织竞争神经网络能够识别成组的相似向量，常用于进行模式分类。

1）自组织竞争神经网络的形成

在生物神经系统中，存在一种“侧抑制”的现象，即一个神经细胞兴奋后，通过它的分支会对周围其他神经细胞产生抑制。这种侧抑制式神经细胞之间出现竞争，虽然开始阶段各个神经细胞都处于程度不同的兴奋状态，由于侧抑制的作用，各细胞之间相互竞争的最终结果是：兴奋作用最强的神经细胞所产生的抑制作用战胜了它周围所有其他细胞的抑制作用而“赢”了，其周围的其他神经细胞则全“输”了。

自组织竞争人工神经网络正是基于上述生物结构和现象形成的。它是一种以无教师示教的方式进行网络训练的，具有自组织功能的神经网络，网络通过自身训练，自动对输入模式进行分类。在网络结构上，自组织竞争人工神经网络一般是由输入层和竞争层构成的单层网络。网络没有隐含层，两层之间各神经元实现双向连接，有时竞争层各神经元之间还存在

横向连接。在学习算法上，它模拟生物神经系统依靠神经元之间的兴奋、协调与抑制、竞争的作用来进行信息处理的动力学原理，指导网络的学习与工作。

2）自组织竞争神经网络的结构

自组织竞争学习网络由输入层和竞争层组成。输入层由接收输入模式的处理单元构成；竞争层的竞争单元争相响应输入模式，胜者表示输入模式的所属类别。输入层单元到竞争层单元的连接为全互连方式，连接权是可调节的。对于给定的一个输入模式，只调节获胜单元的连接权。图 1-25 所示为一个单层自组织竞争学习网络。

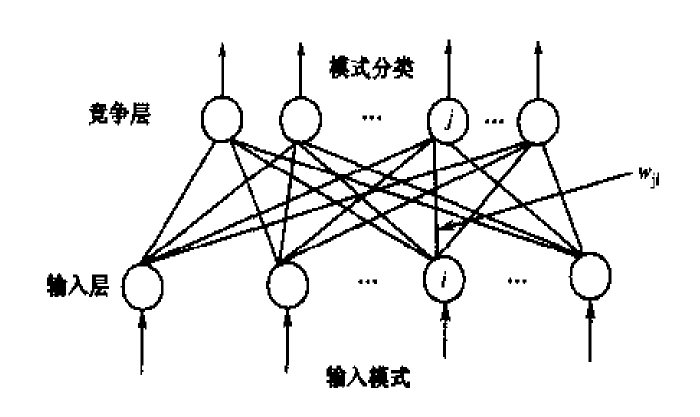


图 1-25 自组织竞争学习网络

3）竞争学习机理

自组织竞争学习网络中，每个竞争单元和输入层单元都有一个连接权，其取值在 0 与 1 之间。为了简化网络，假设任意一个给定竞争单元的权值和总是为 1，即

$$\sum_i w_{ji} \approx 1 \tag{1-40}$$

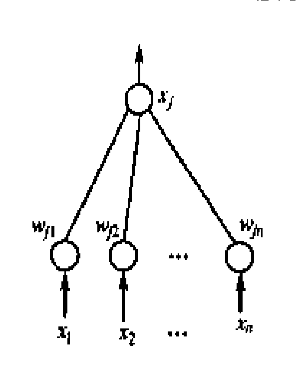


图 1-26 竞争层处理单元

网络学习时，初始权值一般满足上式的一组小的随机数；输入模式是二进制数的 0/1 向量。图 1-26 所示为竞争层的一个处理单元。

竞争单元的处理分为两步，首先计算每个单元输入的加权和，然后在竞争中产生输出。对于第 j 个竞争单元，其输出总和为

$$S_j = \sum_i w_{ji} x_i \tag{1-41}$$

当竞争层所有单元的输入总和计算完毕，便开始竞争。根据“胜者为王，败者为寇”的道理，竞争层中具有最高输入总和的单元被确定为胜者，其输出状态为 1，其他各单元状态为 0，即

$$x_j^c = \begin{cases} 1, S_j > \max(S_k, k \neq j, k = 1,2,\cdots,n) \\ 0, 其他 \end{cases} \tag{1-42}$$

式中，x_j^c表示竞争层第j个单元输出状态。特别地，当$S_j = S_k(k \neq j)$时，通常取位于j左边的处理单元状态为1。

对于某一输入模式，当竞争胜者单元被确定后，更新权值，也只有获胜单元的权值被增加一个量，使得再次遇到该输入模式时，该单元有更大的输入总和。权值更新规则表示为

$$\Delta w_{ji} = \eta\left(\frac{x_i}{m} - w_{ji}\right) \tag{1-43}$$

式中，η 表示学习因子，$0<\eta<1$，反映权值更新速率，一般取值为 0.01～0.3；m 表示输入层状态为1的单元个数；各单元初始权值选其和为1的一组随机数。

竞争学习的基本思想：竞争获胜单元的权值修正，当获胜单元的输入状态为1，相应权值增加；当相连输入单元状态为0，相应权值减少。学习过程中，权值越来越接近于相应的输入状态，这个变化可由如图1-27所示例子学习来反映。图中，第一、第五、第六个输入单元权值不断增大，其他权值不断减小。如果相同的输入模式立刻再次送给网络，那么上一次权值修正的结果将使获胜单元输入总和稍微变大。另外，类似于当前输入的输入模式，也将使相应获胜单元产生较大的输入总和。因此，当用相同或类似模式重复学习时，原已获胜单元有可能再次获胜。

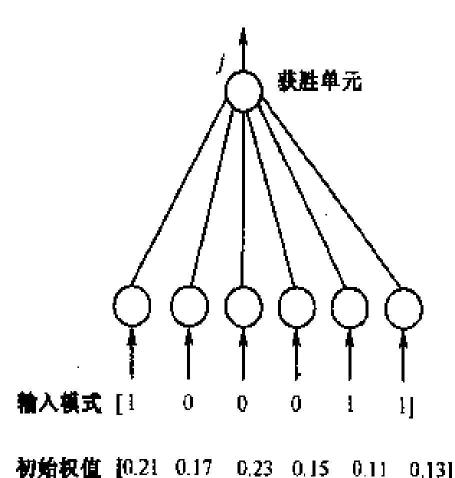


图1-27 竞争学习中修正权值

按照权值修正算式（1-43），获胜单元的一些权值减小一个量，另一些则增大一个量，其结果是获胜单元的权值之和仍然满足为1的约数。原因是将式（1-43）权值修正量对所有输入求和，结果为0。

$$\sum_i \Delta w_{ji} = \eta\left(\frac{1}{m}\sum_i x_i - \sum_i w_{ji}\right) = \eta\ (1-1) = 0$$

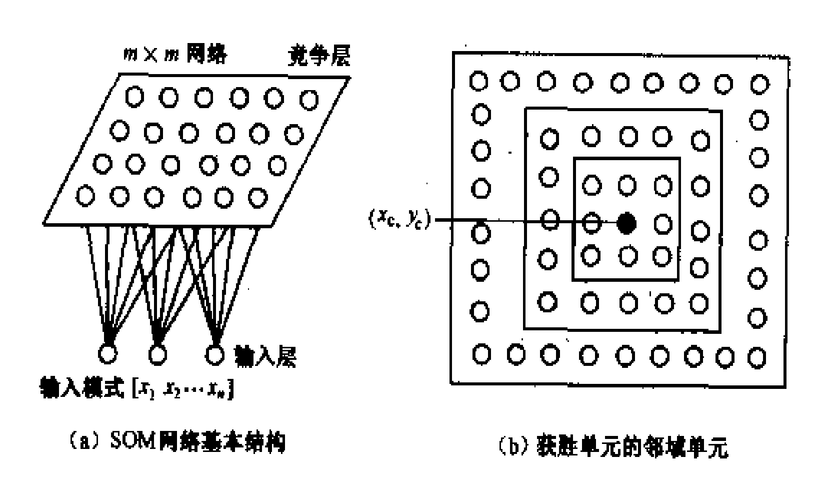
2. 自组织特征映射网络（SOM）

自组织特征映射网络（Self-Organizing feature Map，SOM网络）是由芬兰赫尔辛基大学神经网络专家Kohonen教授在1981年提出的。他认为一个神经网络接收外界输入模式时，将会分为不同的区域，各区域对输入模式具有不同的响应特征，同时这一过程是自动完成的。各神经元的连接权值具有一定的分布。最邻近的神经元互相刺激，而较远一些的则具有较弱的刺激作用。这种网络模拟大脑神经系统自组织特征映射的功能，它是一种竞争式学习网络，在学习中能无监督地进行自组织学习。自组织特征映射神经网络根据输入向量在输入空间的分布情况对它们进行分类。与自组织竞争网络不同的是，在自组织特征映射神经网络中邻近的神经元能够识别输入空间中邻近的部分。这样，自组织特征映射神经网络不但能够

像自组织竞争神经网络一样学习输入的分布情况，而且可以学习进行训练输入向量的拓扑结构。总之，自组织特征映射网络是一种无教师的聚类方法，与传统的模式聚类方法相比，它所形成的聚类中心能映射到一个曲面或平面上，且保持拓扑结构不变。自组织特征映射网络应用广泛，可用于语言识别、图像压缩、机器人控制、优化问题等。

1）自组织特征映射网络的结构

自组织特征映射网络（SOM）也是由输入层和竞争层组成的。与自组织竞争学习网络不同之处是其竞争层按二维网络阵列方式组织，而且权值更新的策略也不同。如图 1-28（a）所示，输入层神经元数为 n，竞争层由 $M=m^2$ 个神经元组成，且构成一个二维平面阵列。输入层与竞争层之间实行全互连接，有时竞争层各神经元之间还实行侧抑制连接。网络中有两种连接权，一种是神经元对外部输入反应的连接权值，另一种是神经元之间的连接权值，它的大小控制着神经元之间的交互作用的大小。对于给定的输入模式，训练过程不仅要调节竞争获胜单元的各连接权值，而且还要调节获胜单元的邻域单元权值。假设竞争获胜单元位于（x_c,y_c）处，则该获胜单元的邻域单元定义为包括在下列矩形框内的所有单元。其中，d 表示由中心点（x_c,y_c）到邻域单元位置（x,y）的距离。图 1-28（b）中标注了 d 为 1,2,3 时获胜单元的邻域单元。



（a）SOM 网络基本结构　　（b）获胜单元的邻域单元

图 1-28　SOM 网络结构

2）自组织特征映射网络的学习机理

自组织特征映射算法是一种无教师示教的聚类方法，它能将任意输入模式在输出层映射成一维或二维离散图形，并保持其拓扑结构不变，即在无教师示教的情况下，通过对输入模式的自组织学习，在竞争层将分类结果表示出来。此外，网络通过对输入模式的反复学习，可以使连接权矢量空间分布密度与输入模式的概率分布趋于一致，即连接权矢量空间分布能反映输入模式的统计特征。

自组织特征映射网络的竞争学习过程包括三个关键点：第一，对于给定输入模式，确定竞争层获胜单元；第二，按照学习规则修正获胜单元及其邻域单元的连接权值；第三，逐渐减少邻域及学习过程中权值的变化量。竞争获胜单元及其邻域单元权值更新规则可表达为

$$w_{ji}(k+1) = w_{ji}(k) + \Delta w_{ji} \tag{1-44}$$

且

$$\Delta w_{ji} = \begin{cases} \eta_c(x_i - w_{ji}), & \text{获胜单元} \\ \eta_N(x_i - w_{ji}), & \text{获胜邻域单元} \end{cases}$$

式中，η_c, η_N 分别表示获胜单元及其邻域单元的权值学习因子，取值在（0,1）区间，且 $\eta_c > \eta_N$；x_i, w_{ji} 表示竞争获胜单元的输入及连接权。

特别要强调的是，在学习过程中，权值学习因子 η_c 及竞争层获胜单元的邻域 Nc 的大小是逐渐减小的。学习因子 η_N 的初值一般取得比较大，随着学习过程的迭代 η_N 值逐渐减小，常用的一个调整策略表达式为

$$\eta_N = \eta_{N_0}\left(1 - \frac{t}{T}\right) \tag{1-45}$$

式中，η_{N_0} 表示 η_N 初始值，通常在 0.2～0.5 之间取值；t 表示迭代次数；T 表示整个迭代设定次数。

邻域的宽度也是随着学习过程的迭代而减少，因此，由竞争获胜单元到邻域单元的距离 d 相应减小。假设 d 的初值记 d_0，一般取值为 1/2 或 1/3 竞争层宽度，d 的减小策略可表达为

$$d = d_0\left(1 - \frac{t}{T}\right) \tag{1-46}$$

自组织映射网络的学习使竞争获胜单元的邻域单元受到激励，邻域之外较远的单元受到抑制。

3. 反传网络（CPN）

反传网络的结构如图 1-29 所示。从形式上看，CPN 也是一个多层前向网络，可用于实现模式映射。这一点与 BP 网络相似。但是在工作机理方面，二者有明显的差异。CPN 除了输入层外，还有两个功能层：第一层为竞争层，其权值学习采用竞争学习策略；第二层为功能输出层，又称 Grossberg 层，它根据竞争层获胜单元的输出，产生一输出模式，采用有导师学习策略（如δ学习规则，Grossberg 学习算法）修正权值，以获得期望输出。之所以这种网络叫反传网络，是当它用做联想记忆时由网络的组织得来的。如图 1-29（b）所示，假设输入模式由向量 X 和向量 Y 组成，即（X,Y）；期望输出模式仍为（X,Y）；网络实际输出模式为（X',Y'）；当用做联想记忆时，如给定输入为（X,*），则网络能回忆出（X,Y）；相反，若给定输入为（*,Y），则能回忆出（X,Y）。

图 1-30 所示为一个 CPN 对于给定的输入模式的一次训练。给定输入模式，竞争层单元竞争响应，只有一个单元获胜，输出状态为 1，其余为 0。如图 1-30（a）、（b）所示，竞争获胜单元激活输出层产生一个输出模式。如图 1-30（c）所示，竞争层、输出层的权值采用不同学习策略调节。

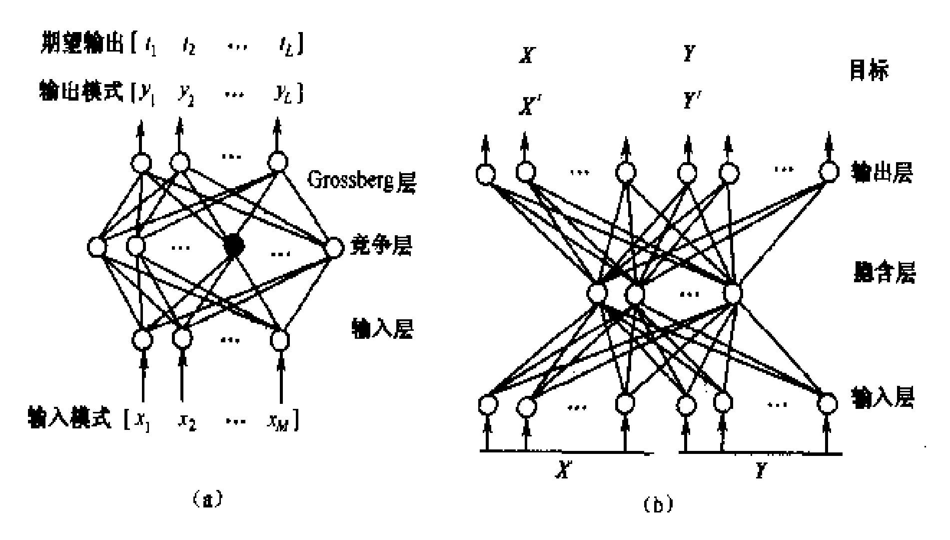


图 1-29 CPN 网络结构

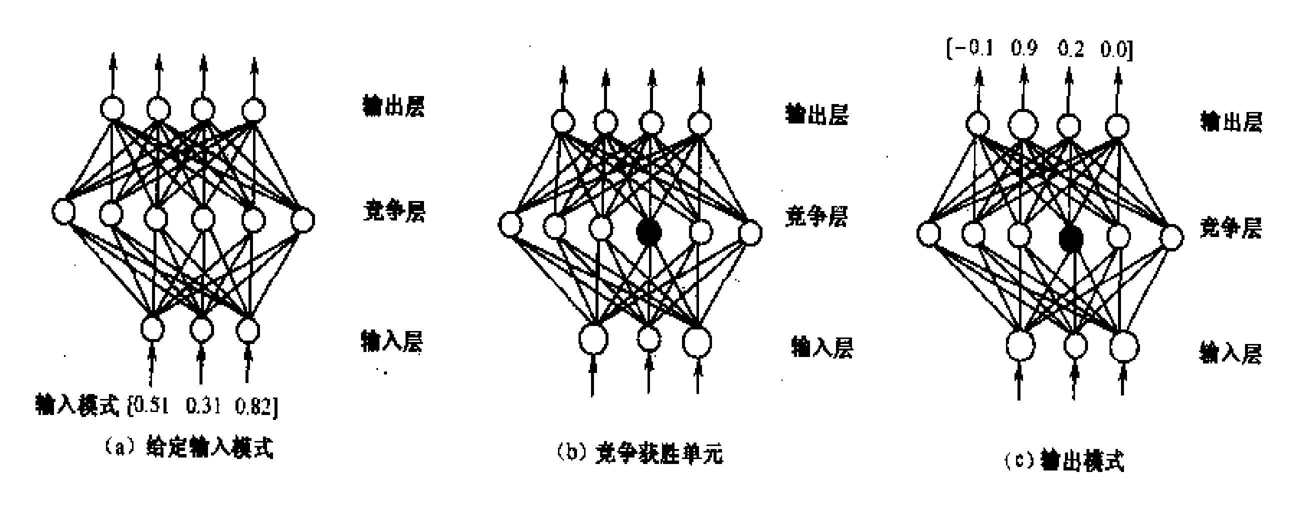


图 1-30 CPN 的训练

4. 自适应共振理论（ART）

自适应共振理论（Adaptive Resonance Theory，ART）是由美国学者 S.Grossberg 和 G.Carpenter 在 20 世纪 80 年代提出的，它的最初模型为 ART-1，只用于二进制数输入，后来有了可适用于连续信号输入的 ART-2，然后又发展了 ART-3。ART-3 是在 ART-2 的基础上发展起来的。它兼容了前两代的功能，同时把两层网络扩大为任意多层。ART-3 是当前最复杂的神经网络之一。以下仅重点介绍 ART-1。

对于 BP 网络，如果学习所用的模式是已知，且是固定的，那么网络经过反复学习可以记住这些固定模式。一旦出现新的模式，网络的学习往往会修改甚至删除已学习的结果，只记得最新的模式。

ART 网络是一种向量模式的识别器，它根据存储的模式对输入向量进行分类，其简化的结构如图 1-31 所示。当存储的模式中有模式和输入模式相匹配时，代表该存储模式的参数就被调整以更接近输入模式。反之，如果在存储模式中，没有发现和输入模式相匹配的模

式时，则输入模式作为新的模式被存储到网络中，其他的存储模式保持不变。ART 网络由比较和识别两层神经元组成，增益控制 1、2 和复置用来控制网络的学习和分类。

ART 网络的工作过程如下。当没有输入时，x 的所有成分均为 0，使得增益矩阵 2 的信号均为 0，因此也使得识别层的输出全为 0。当加入输入向量 x 时，因为 x 必含有不为 0 的元素，因此增益控制 1 和 2 的信号均为 1，从而使比较层的输出向量 C 和输入向量 x 完全相同。接着，识别层寻找和 C 最匹配的神经元 j，并使其输出为 1，即 r_j=1，其他神经元输出均为 0。然后由 r_j=1 决定比较层中对应的存储模式 $T_j=\{t_{j1}, t_{j2}, \cdots, t_{jm}\}$作为向量 P。由于 r_j=1，增益控制 1 的信号被强制为 0，这时比较层的输出就是由比较 x 和 P 产生的结果。如果由上而下的反馈信号和输入模式不匹配（P 和 x 对应的成分不同）时，C 的相应成分就变为 0。因此若 C 的成分中 0 多，而 x 的成分中 1 多时，表明由上而下的反馈模式 P 不是所寻找的模式，此时产生复置信号，使识别层被激活的神经元复原，即使其输出为 0。

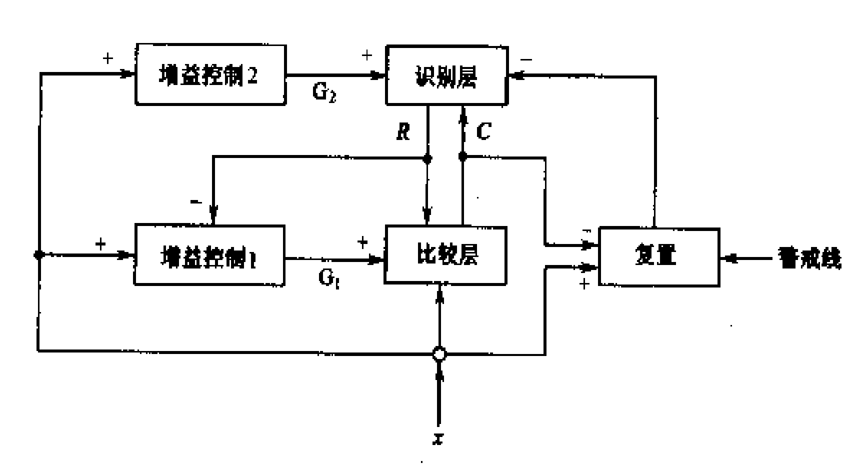


图 1-31　ART 网络的简化结构

如果没有产生复置信号，则表明由上而下的反馈模式和输入模式匹配。反之，则必须搜索其他存储模式，看是否和输入模式相匹配，这一过程反复进行，直到找到一个相匹配的存储模式。若在所有的存储模式中都没有找到相匹配的模式时，则输入模式作为新的模式存储到网络中。

自适应共振理论（ART）网络具有不同的版本。图 1-32 为 ART-1 版本，用于处理二元输入。新的版本，如 ART-2，能够处理连续值输入。

从如图 1-32 所示可见，一个 ART-1 网络含有两层，一个输入层和一个输出层。这两层完全互连，该连接沿着正向（自底向上）和反馈（自顶向下）两个方向进行。自底向上连接至一个输出神经元 i 的权矢量 w_i 形成它所表示的类的一个样本。全部权矢量 w_i 构成网络的长期存储器，用于选择优胜

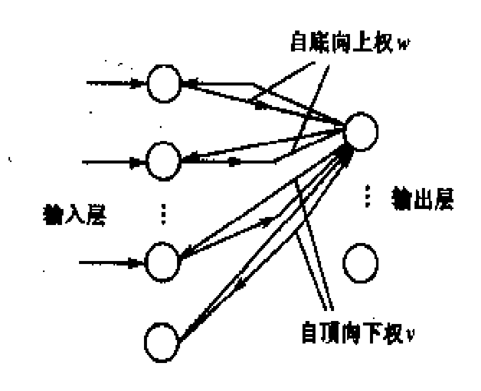


图 1-32　ART-1 网络

的神经元，该神经元的权矢量 w_i 最相似于当前输入模式。自顶向下从一个输出神经元 i 连接的权矢量 v_i 用于警戒测试，即检验某个输入模式是否足够靠近已存储的样本。警戒矢量 v_i 构成网络的短期存储器。v_i 和 w_i 是相关的，w_i 是 v_i 的一个格化副本，即

$$w_i = \frac{v_i}{\varepsilon + \sum v_{ji}} \tag{1-47}$$

在式（1-47）中，ε 为一小的常数；v_{ji} 为 v_i 的第 j 个分量（即从输出神经元 i 到输入神经元 j 连接的权值）。

当 ART-1 网络工作时，其训练是连续进行的，且包括下列步骤：

（1）对于所有输出神经元，预置样本矢量 w_i 及警戒矢量 v_i 的初值，设定每个 v_i 的所有分量为 1，并据式（1-47）计算 w_i。如果一个输出神经元的全部警戒权值均置 1，则称为独立神经元，原因是它不被指定表示任何模式类型。

（2）给出一个新的输入模式 x。

（3）使所有的输出神经元能够参加激发竞争。

（4）从竞争神经元中找到获胜的输出神经元，即这个神经元的 $x*w_i$ 值为最大；在开始训练时或不存在更好的输出神经元时，优胜神经元可能是一个独立神经元。

（5）检查该输入模式 x 是否与获胜神经元的警戒矢量 v_i 足够相似。相似性是由 x 的微分式 r 检测的，即

$$r = \frac{xv_i}{\sum x_i} \tag{1-48}$$

如果 r 值小于警戒阈值 $\rho\,(0<\rho<1)$，那么可以认为 x 与 v_i 是足够相似的。

（6）如果 $r \geqslant \rho$，即存在共振，则转向第（7）步；否则，使获胜神经元暂时无力进一步竞争，并转向第（4）步，重复这一过程直至不存在更多的有能力的神经元为止。

（7）调整最新获胜神经元的警戒矢量 v_i，对它逻辑加上 x，删去 v_i 内而不出现在 x 内的位，根据式（1-48），用新的 v_i 计算自底向上样本矢量 w_i，激活该获胜神经元。

（8）转向第（2）步。

上述训练步骤能够做到，如果同样次序的训练模式被重复地送至此网络，那么其长期和短期存储器保持不变，即该网络是稳定的。假定存在足够多的输出神经元来表示所有不同的类，那么新的模式总是能够学会。因为如果它不与原来存储的样本很好匹配的话（即该网络是塑性的），新模式可被指定给独立输出神经元。

总的说来，ART 的特点如下：

（1）ART 是一种非监督，向量聚类的竞争学习算法；

（2）ART 对“弹性–稳定性”问题提供一种解决办法；

（3）ART 是以认知学和行为学为基础的模型；

（4）ART 在输入与输出层使用大量反馈连接；

（5）ART 可用非线性微分方程组描述；

（6）ART 能工作于实数模式或二值模式输入。

5. 竞争学习网络的特征

竞争学习网络的主要特征表现在竞争层，它采用无导师学习策略，每个竞争单元都相当于一个特征分类器。模式分类是这种网络的重要功能。在竞争学习方案中，网络通过极小化簇内模式距离及极大化不同簇间的距离实现模式分类。注意，这里所说的模式距离是指海明距离（海明距离定义为两个向量中不相同的元素的个数），如模式 010 与 101 的海明距离为 3。通常，竞争学习网络的分簇响应结果与初始权值的设置以及输入模式的组织有一定的关系。可以设想，用一组输入模式来训练网络，网络将输入模式按其海明距离自然分成三类，如图 1-33 所示，假如竞争层的初始权值都是相同的，那么竞争分簇的结果是：首先训练的模式属于类 1，由竞争单元 1 表示；随后训练的模式如果不属于类 1，它就使竞争单元 2 表示类 2；当然，剩下的不属于前两类的模式使单元 3 获胜，为类 3。假如不改变初始权值分布，只改变模式训练的次序，或者不改变上述训练次序，只改变初始权值分布，这两种情况都可能使竞争层单元对分簇响应不一样。对第一种情况，竞争单元 1 获胜，表示类 1；此时竞争单元 1 获胜，可能代表的是类 2 或类 3。有时，如果输入模式组织的不好，那么分类学习可能不稳定。会出现对同一输入模式，先由某一单元响应，以后又由另一单元响应，训练过程中就这样跳来跳去。因此，在竞争学习网络训练时要注意这一关系。

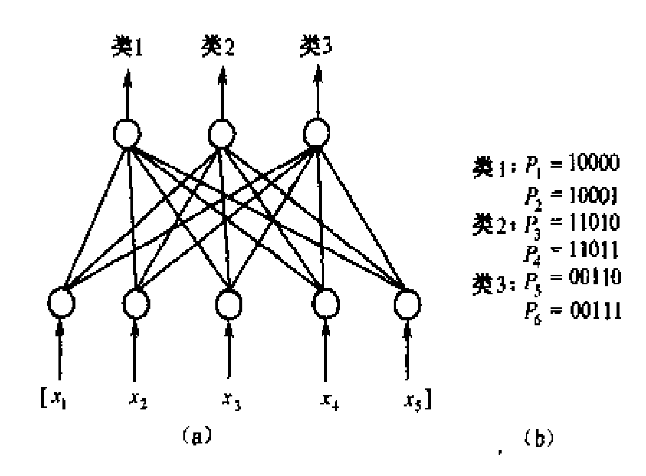


图 1-33　竞争学习聚类分析

竞争学习网络所实现的模式分类情况与典型的 BP 网络分类有所不同。BP 网络分类学习必须知道要将给定模式分成几类；而竞争网络能将给定的模式分成几类预先是不知道的，只有在学习以后才能知道，这种分类能力在许多场合是很有用的；从模式映射能力来看，像 CPN 这样的竞争网络，由于其竞争层仅有一个输出为 1 的获胜单元，因此不能得到某些映射所要求的复杂内部表示；BP 网络能够在最小均方意义上实现输入/输出映射的最优逼近。

竞争学习网络存在一些性能局限。首先，只用部分输入模式训练网络，当用一个明显不同的新的输入模式进行分类时，网络的分类能力可能降低。这是竞争学习采用非推理方式调节权值的缘故。另外，竞争学习对模式变换不冗余，其分类不是大小和旋转不变的，原因是竞争学习网络没有从结构上支持大小和旋转不变的模式分类。

例 1-6　给定一个竞争网络，如图 1-34（a）所示，要求通过训练将输入模式集划分为两大类。设输入模式集为

$$(101)=\boldsymbol{P}_1,\ (100)=\boldsymbol{P}_2,\ (010)=\boldsymbol{P}_3,\ (011)=\boldsymbol{P}_4$$

解：竞争网络是如何将输入模式分成不同类呢？下面先分析一下训练集内四个模式相似性。观察每两个模式之间的海明距离——两个二进制数输入模式不同状态的个数，得到以下关系矩阵。

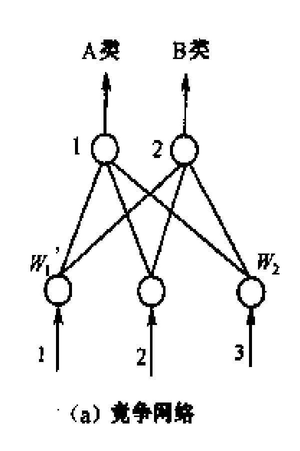
$$\boldsymbol{P}=(p_{ij})=\begin{bmatrix}0&1&3&2\\1&0&2&3\\3&2&0&1\\2&3&1&0\end{bmatrix}$$

所谓两个模式彼此相似，是指海明距离小于某个常量。这里，模式 $\boldsymbol{P}_1$, $\boldsymbol{P}_2$ 彼此相似，$\boldsymbol{P}_3$, $\boldsymbol{P}_4$ 彼此相似；前两个模式 $\boldsymbol{P}_1$, $\boldsymbol{P}_2$ 与后两个模式 $\boldsymbol{P}_3$, $\boldsymbol{P}_4$ 的海明距离较大。因此，输入模式自然分成两类。该网络训练完后得到如下两类

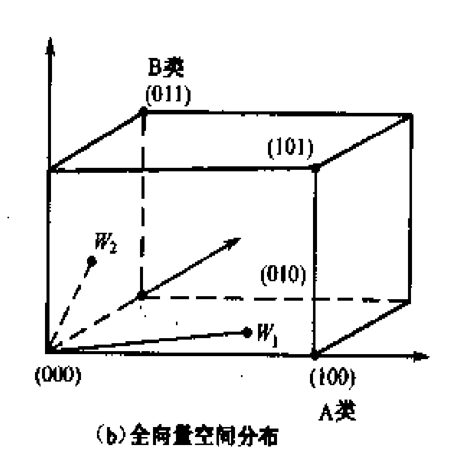
A 类：$\boldsymbol{P}_1=(101), \boldsymbol{P}_2=(100)$；　　　　B 类：$\boldsymbol{P}_3=(010), \boldsymbol{P}_4=(011)$。

每一类包含两个输入模式，同一类模式海明距离为 1，不同类模式的海明距离为 2 或 3。网络的分类原则来源于输入模式的固有特征。用不同的初始权值反复进行训练，网络便能自组织学习，完成正确的模式分组。

图 1-34（b）为训练完成后的权向量及训练集内输入模式的空间分布。所有 3 输入二进制向量位于三维立方体的各顶点，竞争层单元 1 的权向量最接近于 A 类的两个模式；竞争单元 2 的权向量最接近于 B 类的两个模式。若输入模式为 A 类模式，则单元 1 竞争获胜；同样，当输入为 B 类的两个模式时，单元 2 竞争获胜。图中，权向量相对比较短，这是由单元权值之和必须为 1 约束的结果。



(a) 竞争网络



(b) 全向量空间分布

图 1-34　竞争分类

1.2.7　学习向量量化（LVQ）神经网络

学习向量量化神经网络是一种由输入层、竞争层和线性输出层组成的混合网络，LVQ 神经网络的学习结合了竞争学习和有监督学习来形成分类。学习规则为

$$\Delta w_{ij}(k)=\begin{cases}\eta(x_j-w_{ij}), & \text{若神经元}j\text{竞争获胜}\\ 0, & \text{若神经元}j\text{竞争失败}\end{cases} \tag{1-49}$$

在 LVQ 神经网络中，第一层的每个神经元都指定给某个类，常常几个神经元被指定给同一个类。每类再被指定给第二层的一个神经元。第一层的神经元的个数与第二层的神经元个数至少相同，并且通常要大一些。和竞争网络一样，LVQ 神经网络的第一层的每个神经元学习原型向量，它可以对输入空间的区域分类。然而，不是通过计算内积得到输入和权值向量中最接近者，而是通过直接计算距离的方法来模拟 LVQ 网络。直接计算距离的一个优点是向量不必先格式化，当向量规格化了，无论是采用计算内积的方法还是直接计算距离，网络的响应将是相同的。

LVQ 神经网络与竞争网络的特性几乎相同（至少对规格化向量）。然而，对于竞争网络，有非零输出的神经元表示输入向量属于那个类。而对于 LVQ 神经网络，竞争获胜的神经元表示的是一个子类而非一个类。一个类可能由几个不同的神经元（子类）组成。

图 1-35 给出一个学习向量量化（LVQ）神经网络。它是由输入层、竞争层（又称隐含层）和输出层组成的。该网络在输入层和竞争层间为完全连接，而在竞争层与输出层间为部分连接，每个输出神经元与竞争神经元的不同组相连接。竞争层和输出层间的连接权值固定为 1。输入层和竞争层间的连接权值建立参考向量的分量（对每个竞争神经元指定一个参考向量）。在网络的训练过程中，这些权值被修改。竞争神经元（又称隐含神经元）和输出神经元都具有二进制数输出值。当某个输入模式送至网络时，参考向量最接近输入模式的竞争神经元因获得激发而赢得竞争，因而允许它产生一个“1”。其他竞争神经元都被迫产生“0”。与包含获胜神经元的竞争神经元组成相连接的输出神经元也发出“1”，而其他输出神经元均发出“0”。产生“1”的输出神经元给出输入模式的类，每个输出神经元被表示为不同的类。

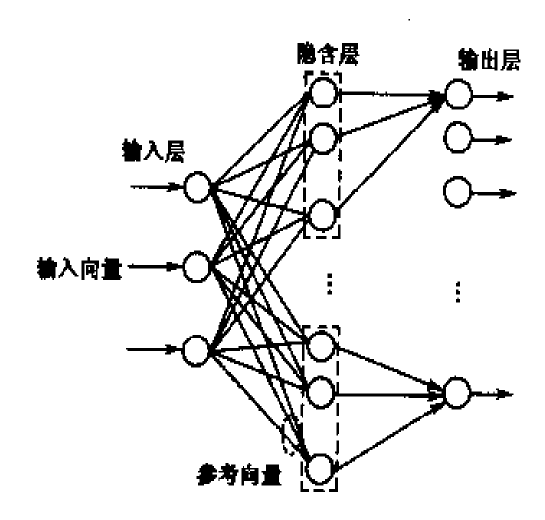


图 1-35 学习向量量化（LVQ）神经网络

最简单的 LVQ 神经网络训练步骤如下：

（1）预置参考向量的初始权值；

（2）提供给网络一个训练输入模式；

（3）计算输入模式与每个参考向量间的 Euclidean 距离；

（4）更新最接近输入模式的参考向量（即获胜竞争神经元的参考向量）的权值。如果获胜竞争神经元以输入模式一样的类属于连接至输出神经元的缓冲器，那么参考向量应更接

近输入模式；否则，参考向量就离开输入模式。

（5）返回（2），以某个新的训练输入模式重复本过程，直至全部训练模式被正确地分类或者满足某个终止准则为止。

1.2.8 Elman 神经网络

Elman 神经网络包含一个双曲正切 S 型隐含层和一个线性输出层，S 型隐含层接收网络输入和自身的反馈，线性输出层从 S 型隐含层得到输入。它是反馈网络中最有代表性的例子，这种网络也有多层结构，如图 1-36 所示，在这种网络中，除了普通的隐含层外，还有一个特别的隐含层，有时称为上下文层或状态层。该层从普通隐含层接收反馈信号，上下文层内的神经元输出被前向传输至隐含层。Elman 神经网络的这种组合结构特点使得其能在有限的时间内以任意精度逼近任意函数。需要通过给递归层设置足够的神经元来实现。当需要逼近的函数复杂性增加时，所需的递归层神经元数目也要增加。由于 Elman 网络是 S 型/线性（Sigmoid/Linear）网络，它能够表达含有有限个不连续点的函数。又因为它们有一个反馈连接，所以它被训练后不仅能够识别和产生空间模式，还能够识别和产生时间模式。

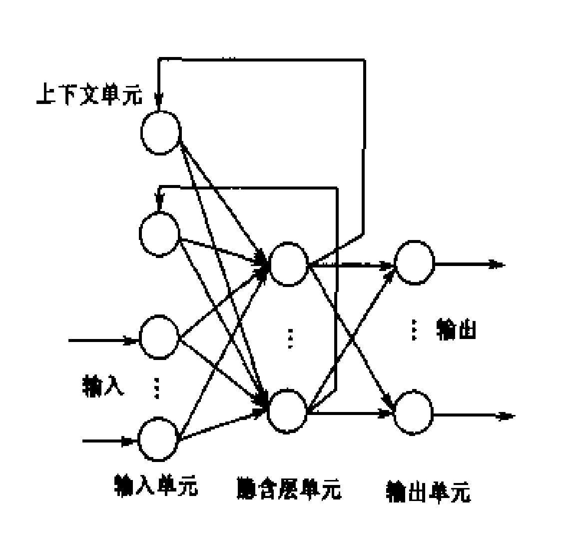


图 1-36 Elman 神经网络

对于多输入多输出网络，设上下文层的输出为 $\boldsymbol{y}_c(k)$，隐含层的输入和输出分别为 $\boldsymbol{x}_0(k)$ 和 $\boldsymbol{o}(k)$，网络在外部输入时间序列 $\boldsymbol{x}(k)$ 作用下的网络输出序列为 $\boldsymbol{y}(k)$，则有

$$\begin{aligned}\boldsymbol{x}_0(k+1)&=\boldsymbol{W}_1\boldsymbol{y}_c(k+1)+\boldsymbol{W}\boldsymbol{x}(k)+\boldsymbol{\theta}_1\\ \boldsymbol{y}_c(k)&=\boldsymbol{o}(k-1)=f(\boldsymbol{x}_0(k-1))\\ \boldsymbol{y}(k)&=\boldsymbol{W}_2\boldsymbol{o}(k)+\boldsymbol{\theta}_2\end{aligned} \tag{1-50}$$

式中，$\boldsymbol{W}_1$ 为输入层与隐含层间的连接权值；$\boldsymbol{W}_2$ 为隐含层与输出层间的连接权值；$f(\bullet)$ 为 S 型激活函数。

当 Elman 神经网络的上下文层存在增益为 α 的自反馈连接时，称为改进型 Elman 神经网络。此时，网络能模拟更高阶的动态系统，其上下文层的输出 $\boldsymbol{y}_c(k)$ 变为

$$\boldsymbol{y}_c(k)=\boldsymbol{o}(k-1)+\alpha\boldsymbol{y}_c(k-1) \tag{1-51}$$

如果 Elman 神经网络只有正向连接是适用的，而反馈连接被预定为恒值，那么这些网络可视为普通的前馈网络。Elman 神经网络可以用 BP 算法进行训练；否则，可采用遗传算法。

1.2.9 Hopfield 神经网络

Hopfield 神经网络是 1982 年美国物理学家 Hopfield 首先提出来的。前面讨论的 BP 前向神经网络的特点是网络中没有反馈连接，不考虑输出与输入间在时间上的滞后影响，其输出与输入间仅是一种映射关系。而 Hopfield 网络则不同，它采用反馈连接，考虑输出与输入间在时间上的传输延迟，所表示的是一个动态过程，需要用差分方程或微分方程来描述，因而 Hopfield 网络是一种由非线性元件构成的反馈系统，其稳定状态的分析比 BP 网络要复杂得多。此外，在网络的学习训练，即加权系数的调整方面也不同，BP 前向网络采用的是一种有监督的误差均方修正方法，学习计算过程较长，收敛速度较慢。而 Hopfield 网络则是采用有监督的 Hebb 规则（用输入模式作为目标模式）来设计连接权。在一般情况下，计算的收敛速度很快。该网络主要用于联想记忆和优化计算。

在 Hopfield 网络中，每一个神经元都和所有其他神经元相连接，故又称为全互联网络。研究表明，当连接加权系数矩阵无自连接并具有对称性质，即

$$w_{ii}=0\,,\ w_{ij}=w_{ji}\quad (i\neq j)\qquad (i,j=1,2,\ \cdots,n)$$

时，算法是收敛的。

根据网络的输出是离散量或是连续量，Hopfield 网络分为离散型和连续型两种。下面分别对它们进行讨论。

1．离散型 Hopfield 网络

1）网络的结构和工作方式

离散型 Hopfield 网络的结构如图 1-37 所示。

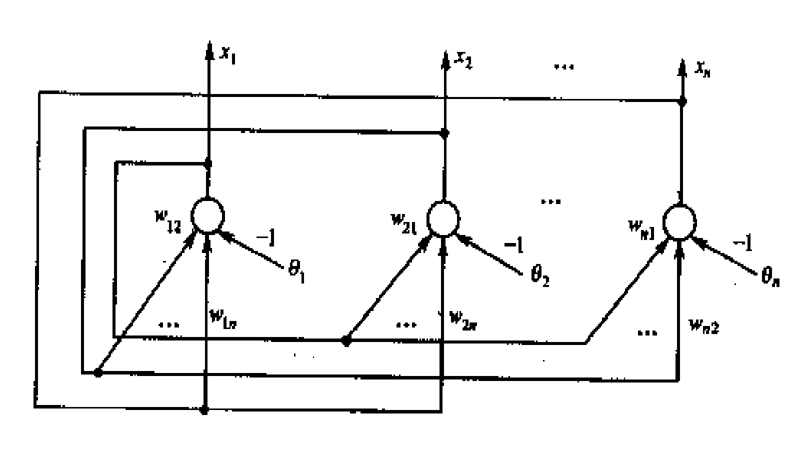


图 1-37 离散型 Hopfield 网络的结构

由图 1-37 可以看出，它是一个单层网络，共有 n 个神经元节点，每个节点输出均连接到其他神经元的输入，同时所有其他神经元的输出均连到该神经元的输入。对于每一个神经元节点，其工作方式仍同以前一样，即

$$\begin{cases} s_i(k)=\sum_{j=1}^{n} w_{ij}x_j(k)-\theta_i \\ x_i(k+1)=f(s_i(k)) \end{cases} \tag{1-52}$$

式中，$w_{ii}=0$；θ_i为阈值；$f(\bullet)$是变换函数。对于离散 Hopfield 网络，$f(\bullet)$通常取为二值函数，即

$$f(s)=\begin{cases}1, s\geqslant 0\\ -1, s<0\end{cases} \quad 或 \quad f(s)=\begin{cases}1, s\geqslant 0\\ 0, s<0\end{cases} \tag{1-53}$$

整个网络有如下两种工作方式。

（1）异步方式或串行工作方式

在某一时刻只有一个神经元按照式（1-52）改变状态，而其余神经元的输出保持不变，这一变化的神经元可以按照随机方式或预定的顺序来选择。例如，若达到的神经元为第 i 个，则有

$$\begin{cases} x_i(k+1)=f(\sum_{j=1}^{n} w_{ij}x_j(k)-\theta_i) \\ x_j(k+1)=x_j(k), \quad j\neq i \end{cases} \tag{1-54}$$

其调整次序可以随机选定，也可按规定的次序进行。

（2）同步方式或并行工作方式

在某一时刻有 $n_1(0<n_1\leqslant n)$个神经元按照式（1-52）改变状态，而其余神经元的输出保持不变。变化的这一组神经元可以按照随机方式或预定的顺序来选择。当 $n_1=n$ 时，称为全并行方式，此时所有神经元都按照式（1-52）改变状态，即

$$x_i(k+1)=f(\sum_{j=1}^{n} w_{ij}x_j(k)-\theta_i),(i=1,2,\cdots,n) \tag{1-55}$$

上述同步计算方式也可写成如下的矩阵形式

$$\boldsymbol{x}(k+1)=\boldsymbol{f}(\boldsymbol{W}\boldsymbol{x}(k)-\boldsymbol{\theta})$$

式中，$\boldsymbol{x}(k)=[x_1(k),x_2(k),\cdots,x_n(k)]^{\mathrm{T}}$ 和 $\boldsymbol{\theta}=[\theta_1,\theta_2,\cdots,\theta_n]^{\mathrm{T}}$ 是向量，W 是由 w_{ij} 所组成的 $n\times n$ 矩阵；$f(s)$是向量函数，它表示 $\boldsymbol{f}(s)=[f(s_1),f(s_2),\cdots,f(s_n)]^{\mathrm{T}}$

该网络是动态的反馈网络，其输入是网络的状态初值

$$\boldsymbol{x}(0)=[x_1(0),x_2(0),\cdots,x_n(0)]^{\mathrm{T}}$$

输出是网络的稳定状态 $\lim_{k\to\infty}\boldsymbol{x}(k)$。

2）稳定性和吸引子

从上述工作中可以看出，离散型 Hopfield 网络实质上是一个离散的非线性动力学系统。如果系统是稳定的，则它可以从任意初态收敛到一个稳定状态；若系统是不稳定的，由于网络节点输出点只有 1 和–1（或 1 和 0）两种状态，因而系统不可能出现无限发散，只可

能出现限幅的自持振荡或极限环。

在 Hopfield 网络的拓扑结构及权值矩阵均一定的情况下，网络的稳定状态将与其初始状态有关。也就是说，Hopfield 网络是一种能储存若干个预先设置的稳定状态的网络。若将稳态视为一个记忆样本，则初态朝稳态的收敛过程便是寻找记忆样本的过程。初态可认为是给定样本的部分信息，网络改变的过程可认为是从部分信息找到全部信息，从而实现了联想记忆的功能。

若将稳态与某种优化计算的目标函数相对应，并作为目标函数的极小点，则初态朝稳态的收敛过程便是优化计算过程。该优化计算是在网络演变过程中自动完成的。

（1）稳定性

若网络的状态 $\boldsymbol{x}$ 满足 $\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{Wx}-\boldsymbol{\theta})$，则称 $\boldsymbol{x}$ 为网络的稳定点或吸引子。

定理 1.3　对于离散型 Hopfield 网络，若按异步方式调整状态，且连接权矩阵 $\boldsymbol{W}$ 为对称阵，即 $w_{ij}=w_{ji}$，则对于任意初态，网络都最终收敛到一个吸引子。

证明：定义网络的能量函数为

$$E(k)=-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}w_{ij}x_j(k)x_i(k)+\sum_{i=1}^{n}\theta_i x_i(k)=-\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}(k)\boldsymbol{W}\boldsymbol{x}(k)+\boldsymbol{x}^{\mathrm{T}}(k)\boldsymbol{\theta}$$

由于神经元节点的状态只能取 1 和−1（或 1 和 0）两种状态，因此上述定义的能量函数 $E(k)$ 是有界的。

令
$$\Delta E(k)=E(k+1)-E(k),\Delta\boldsymbol{x}(k)=\boldsymbol{x}(k+1)-\boldsymbol{x}(k)$$

则

$$\begin{aligned}\Delta E(k)&=E(k+1)-E(k)\\&=-\frac{1}{2}[\boldsymbol{x}(k)+\Delta\boldsymbol{x}(k)]^{\mathrm{T}}\boldsymbol{W}[\boldsymbol{x}(k)+\Delta\boldsymbol{x}(k)]+[\boldsymbol{x}(k)+\Delta\boldsymbol{x}(k)]^{\mathrm{T}}\boldsymbol{\theta}-\\&\quad\left[-\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}(k)\boldsymbol{W}\boldsymbol{x}(k)+\boldsymbol{x}^{\mathrm{T}}(k)\boldsymbol{\theta}\right]\\&=-\Delta\boldsymbol{x}^{\mathrm{T}}(k)\boldsymbol{W}\boldsymbol{x}(k)-\frac{1}{2}\Delta\boldsymbol{x}^{\mathrm{T}}(k)\boldsymbol{W}\Delta\boldsymbol{x}(k)+\Delta\boldsymbol{x}^{\mathrm{T}}(k)\boldsymbol{\theta}\\&=-\Delta\boldsymbol{x}^{\mathrm{T}}(k)[\boldsymbol{W}\boldsymbol{x}(k)-\boldsymbol{\theta}]-\frac{1}{2}\Delta\boldsymbol{x}^{\mathrm{T}}(k)\boldsymbol{W}\Delta\boldsymbol{x}(k)\end{aligned}$$

由于假定异步工作方式，因此可设第 k 时刻只有第 i 个神经元调整状态，即将 $\Delta\boldsymbol{x}(k)=[0\ \cdots\ 0\ \ \Delta x_i(k)\ \ 0\ \cdots\ 0]^{\mathrm{T}}$ 代入上式，则有

$$\Delta E(k)=-\Delta x_i(k)[\sum_{j=1}^{n}w_{ij}x_j(k)-\theta_i]-\frac{1}{2}\Delta x_i^2 w_{ii}$$

令 $s_i(k)=\sum_{j=1}^{n}w_{ij}x_j(k)-\theta_i$

则 $$\Delta E(k)=-\Delta x_i(k)\left[s_i(k)+\frac{1}{2}\Delta x_i(k)w_{ii}\right]=-\Delta x_i(k)s_i(k) \quad (w_{ii}=0)$$

设神经元节点取 1 和−1 两种状态，则

$$x_i(k+1)=f[s_i(k)]=\begin{cases}1 & s_i(k)\geqslant 0\\ -1 & s_i(k)<0\end{cases}$$

下面考虑$\Delta x_i(k)$可能出现的各种情况：

① $x_i(k)=-1, x_i(k+1)=f[s_i(k)]=1$时，有$\Delta x_i(k)=2, s_i(k)\geqslant 0$，则

$$\Delta E(k)\leqslant 0$$

② $x_i(k)=1, x_i(k+1)=f[s_i(k)]=-1$时，有$\Delta x_i(k)=-2, s_i(k)<0$，则

$$\Delta E(k)<0$$

③ $x_i(k+1)=x_i(k)$时，有$\Delta x_i(k)=0$，则

$$\Delta E(k)=0$$

可见，在任何情况下均有$\Delta E(k)\leqslant 0$，由于$E(k)$有下界，因此$E(k)$将收敛到一常数。下面需考察$E(k)$收敛到常数时是否对应于网络的吸引子。根据上述分析，当$\Delta E(k)=0$时，相应于以下两种情况之一。

① $x_i(k+1)=x_i(k)=1$或$x_i(k+1)=x_i(k)=-1$；

② $x_i(k)=-1, x_i(k+1)=1, s_i(k)=0$。

对于情况①，表明x_i已进入稳定状态；对于情况②，网络继续演变时$x_i=1$也将不会再变化，因为若x_i由 1 再变回−1，则有$\Delta E<0$，它与$E(k)$已收敛到常数相矛盾，所以网络最终将收敛到吸引子。

上述分析时假设$w_{ii}=0$，实际上不难看出，当$w_{ii}>0$时，上述结论仍成立，而且收敛过程将更快。

上面证明时假设神经元节点取 1 和−1 两种状态，不难验证x取 1 和 0 两种状态时，上述结论也成立。

定理 1.4 对于离散型 Hopfield 网络，若按同步方式调整状态，且连接权矩阵$\boldsymbol{W}$为非负定对称阵，则对于任意初态，网络都最终收敛到一个吸引子。

证明：前已求得

$$\Delta E(k)=E(k+1)-E(k)=-\Delta \boldsymbol{x}^{\mathrm{T}}(k)[\boldsymbol{W}\boldsymbol{x}(k)-\boldsymbol{\theta}]-\frac{1}{2}\Delta \boldsymbol{x}^{\mathrm{T}}(k)\boldsymbol{W}\Delta \boldsymbol{x}(k)$$

$$=-\Delta \boldsymbol{x}^{\mathrm{T}}(k)\boldsymbol{s}(k)-\frac{1}{2}\Delta \boldsymbol{x}^{\mathrm{T}}(k)\boldsymbol{W}\Delta \boldsymbol{x}(k)=-\sum_{i=1}^{n}\Delta x_i(k)s_i(k)-\frac{1}{2}\Delta \boldsymbol{x}^{\mathrm{T}}(k)\boldsymbol{W}\Delta \boldsymbol{x}(k)$$

前已证得，对于所有的i，有$-\Delta x_i(k)s_i(k)\leqslant 0$，因此只要$\boldsymbol{W}$为非负定阵即有$\Delta E(k)\leqslant 0$，也即$E(k)$最终将收敛到一个常数值，并按照如上面同样的分析可说明网络最终将收敛到吸引子。

可见对于同步方式，它对连接权矩阵 $\boldsymbol{W}$ 的要求更高了，若不满足 $\boldsymbol{W}$ 为非负定对称矩阵的要求，则网络可能出现自持振荡或极限环。

由于异步工作方式比同步工作方式有更好的稳定性能，因此实现时较多采用异步工作方式。异步工作方式的主要缺点是失去了神经网络并行处理的优点。

（2）吸引子的性质

定理 1.5 若 $\boldsymbol{x}$ 是网络的一个吸引子，且对于所有的 i，$\theta_i = 0, \sum_{j=1}^{n} w_{ij}x_i \neq 0$，则 $-\boldsymbol{x}$ 也一定是该网络的吸引子。

证明：由于 $\boldsymbol{x}$ 是吸引子，即 $\boldsymbol{x} = f[\boldsymbol{W}\boldsymbol{x}]$，从而有 $f[\boldsymbol{W}(-\boldsymbol{x})] = f[-\boldsymbol{W}\boldsymbol{x}] = -f[\boldsymbol{W}\boldsymbol{x}] = -\boldsymbol{x}$，即 $-\boldsymbol{x}$ 也是网络的吸引子。

定理 1.6 若 $\boldsymbol{x}^{(a)}$ 是网络的吸引子，则与 $\boldsymbol{x}^{(a)}$ 的海明距离 $d_H(\boldsymbol{x}^{(a)}, \boldsymbol{x}^{(b)}) = 1$ 的 $\boldsymbol{x}^{(b)}$ 一定不是吸引子，海明距离定义为两个向量中不相同的元素的个数。

证明：不失一般性，设 $x_1^{(a)} \neq x_1^{(b)}, x_j^{(a)} = x_j^{(b)} (j = 2, 3, \cdots, n)$。因为 $w_{11} = 0$，所以有

$$x_1^{(a)} = f\left[\sum_{j=2}^{n} w_{1j}x_j^{(a)} - \theta_1\right] = f\left[\sum_{j=2}^{n} w_{1j}x_j^{(b)} - \theta_1\right] \neq x_1^{(b)}$$

所以 $\boldsymbol{x}^{(b)}$ 一定不是网络的吸引子。

推论：若 $\boldsymbol{x}^{(a)}$ 是网络的吸引子，且对于所有的 i，$\theta_i = 0, \sum_{j=1}^{n} w_{ij}x_i \neq 0$，则 $d_H(\boldsymbol{x}^{(a)}, \boldsymbol{x}^{(b)}) = n-1$ 的 $\boldsymbol{x}^{(b)}$ 一定不是吸引子。

证明：若 $d_H(\boldsymbol{x}^{(a)}, \boldsymbol{x}^{(b)}) = n-1$，则 $d_H(-\boldsymbol{x}^{(a)}, \boldsymbol{x}^{(b)}) = 1$。根据定理 1.5，$\boldsymbol{x}^{(a)}$ 是网络的吸引子，$-\boldsymbol{x}^{(a)}$ 也是网络的吸引子，根据定理 1.6，$\boldsymbol{x}^{(b)}$ 一定不是吸引子。

（3）吸引域

为了能实现联想记忆，对于每一个吸引子应该有一定的吸引范围，这个吸引范围便称为吸引域。下面给出较严格的定义。

① 若 $\boldsymbol{x}^{(a)}$ 是吸引子，对于异步方式，若存在一个调整次序可以从 $\boldsymbol{x}$ 演变到 $\boldsymbol{x}^{(a)}$，则称 $\boldsymbol{x}$ 弱吸引到 $\boldsymbol{x}^{(a)}$；若对于任意调整次序都可以从 $\boldsymbol{x}$ 演变到 $\boldsymbol{x}^{(a)}$，则称 $\boldsymbol{x}$ 强吸引到 $\boldsymbol{x}^{(a)}$。

② 对所有 $\boldsymbol{x} \in R(\boldsymbol{x}^{(a)})$ 均有 $\boldsymbol{x}$ 弱（强）吸引到 $\boldsymbol{x}^{(a)}$，则称 $R(\boldsymbol{x}^{(a)})$ 为 $\boldsymbol{x}^{(a)}$ 的弱（强）吸引域。

对于同步方式，由于无调整次序问题，因此，相应的吸引域也无强弱之分。

对于异步方式，对同一个状态，若采用不同的调整次序，则有可能弱吸引到不同的吸引子。

3）连接权的设计

Hopfield 网络没有与之相关的学习规则，它的权值不被训练，也不会自己学习。它的权

值矩阵是事前用基于 Lyapunov 函数的设计过程采用 Hebb 规则来计算出来的。在这种网络中，不断更新的不是权值，而是网络中各神经元的状态，网络演变到稳定时各神经元的状态便是问题的解。为了保证 Hopfield 网络在异步方式工作时能稳定收敛，连接权矩阵 $\boldsymbol{W}$ 应是对称的。若要保证同步方式收敛，则要求 $\boldsymbol{W}$ 为非负定阵，这个要求比较高。因而设计一般只保证异步方式收敛。另外一个要求是对于给定的样本必须是网络的吸引子，而且要有一定的吸引域，这样才能正确实现联想记忆功能。

设给定 m 个样本 $\boldsymbol{x}^{(k)}(k=1,2,\cdots,m)$，为了实现上述功能，通常采用有监督的 Hebb 规则（用输入模式作为目标模式）来设计连接权，连接权可按以下两种情况进行计算。

（1）当网络节点状态为 1 和−1 两种状态，即 $\boldsymbol{x}\in\{-1,1\}^n$ 时，相应的连接权为

$$w_{ij}=\begin{cases}\sum_{k=1}^{m}x_i^{(k)}x_j^{(k)} & i\neq j\\ 0 & i=j\end{cases} \tag{1-56}$$

写成矩阵形式则为

$$\begin{aligned}\boldsymbol{W}&=[\boldsymbol{x}^{(1)}\boldsymbol{x}^{(2)}\cdots\boldsymbol{x}^{(m)}]\begin{bmatrix}\boldsymbol{x}^{(1)\mathrm{T}}\\ \boldsymbol{x}^{(2)\mathrm{T}}\\ \vdots\\ \boldsymbol{x}^{(m)\mathrm{T}}\end{bmatrix}-m\boldsymbol{I}\\ &=\sum_{k=1}^{m}\boldsymbol{x}^{(k)}\boldsymbol{x}^{(k)\mathrm{T}}-m\boldsymbol{I}=\sum_{k=1}^{m}(\boldsymbol{x}^{(k)}\boldsymbol{x}^{(k)\mathrm{T}}-\boldsymbol{I})\end{aligned} \tag{1-57}$$

式中，$\boldsymbol{I}$ 为单位矩阵。

证明： 当输入为单个样本式 $\boldsymbol{x}$ 时，若网络是稳定的，则稳定时输出 $\mathrm{sgn}(\boldsymbol{W}\boldsymbol{x})$ 应该也是 $\boldsymbol{x}$，即应有

$$\boldsymbol{x}=\mathrm{sgn}(\boldsymbol{W}\boldsymbol{x})$$

或

$$x_i=\mathrm{sgn}(\sum_{j=1}^{n}w_{ij}x_j)$$

根据上式，考虑到符号函数的性质，可知 x_i 与 $\sum_{j=1}^{n}w_{ij}x_j$ 同符号，因此有

$$x_i\sum_{j=1}^{n}w_{ij}x_j>0$$

而为了满足上式，权值可按有监督的 Hebb 规则（用输入样本作为目标模式）来设置，即

$$w_{ij}=\eta x_i x_j$$

这是因为，此时满足

$$x_i\sum_{j=1}^{n}w_{ij}x_j=x_i\sum_{j=1}^{n}\eta x_i x_j^2=\eta x_i^2\sum_{j=1}^{n}x_j^2=\eta x_i^2 n>0$$

当输入为 m 个样本 $\boldsymbol{x}^{(k)}(k=1,2,\cdots,m)$时，权值可根据以上直接推广为由下式来设置

$$w_{ij}=\eta\sum_{k=1}^{m}x_i^{(k)}x_j^{(k)}$$

若取 $w_{ij}=0,\eta=1$，则上式可写为

$$w_{ij}=\sum_{\substack{k=1\\i\neq j}}^{m}x_i^{(k)}x_j^{(k)}$$

（2）当网络节点状态为1和0两种状态，即 $\boldsymbol{x}\in\{0,1\}^n$ 时，相应的连接权为

$$w_{ij}=\begin{cases}\sum\limits_{k=1}^{m}(2x_i^{(k)}-1)(2x_j^{(k)}-1) & i\neq j\\ 0 & i=j\end{cases}\tag{1-58}$$

写成矩阵形式则为

$$\boldsymbol{W}=\sum_{k=1}^{m}(2\boldsymbol{x}^{(k)}-\boldsymbol{b})(2\boldsymbol{x}^{(k)}-\boldsymbol{b})^{\mathrm{T}}-m\boldsymbol{I}\tag{1-59}$$

式中，$\boldsymbol{b}=[1\quad 1\quad\cdots\quad 1]^{\mathrm{T}}$。

显然，上面所设计的连接权矩阵满足对称性的要求。

（3）下面进一步分析所给样本是否为网络的吸引子，这一点是十分重要的。下面以 $\boldsymbol{x}\in\{-1,1\}^n$ 的情况为例进行分析。

若 m 个样本 $\boldsymbol{x}^{(k)}(k=1,2,\cdots,m)$ 是两两正交的，即

$$\begin{cases}\boldsymbol{x}^{(i)\mathrm{T}}\boldsymbol{x}^{(j)}=0 & (i\neq j)\\ \boldsymbol{x}^{(i)\mathrm{T}}\boldsymbol{x}^{(i)}=n\end{cases}$$

则有

$$\begin{aligned}\boldsymbol{W}\boldsymbol{x}^{(k)}&=(\sum_{i=1}^{m}\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)\mathrm{T}}-m\boldsymbol{I})\boldsymbol{x}^{(k)}=\sum_{i=1}^{m}\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)\mathrm{T}}\boldsymbol{x}^{(k)}-m\boldsymbol{x}^{(k)}\\&=n\boldsymbol{x}^{(k)}-m\boldsymbol{x}^{(k)}=(n-m)\boldsymbol{x}^{(k)}\end{aligned}$$

可见，只要满足 $n-m>0$，便有

$$f[\boldsymbol{W}\boldsymbol{x}^{(k)}]=f[(n-m)\boldsymbol{x}^{(k)}]=\boldsymbol{x}^{(k)}$$

即 $\boldsymbol{x}^{(k)}$ 是网络的吸引子。

若 m 个样本 $\boldsymbol{x}^{(k)}(k=1,2,\cdots,m)$ 不是两两正交，且设向量之间的内积为 $\boldsymbol{x}^{(i)\mathrm{T}}\boldsymbol{x}^{(j)}=\beta_{ij}$，显然，$\beta_{ii}=n,(i=1,2,\cdots,m)$，则有

$$Wx^{(k)}=\sum_{i=1}^{m}x^{(i)}x^{(i)\mathrm{T}}x^{(k)}-mx^{(k)}=(n-m)x^{(k)}+\sum_{\substack{i=1\\i\neq k}}^{m}x^{(i)}\beta_{ik}$$

取其中第 j 个元素

$$Wx_j^{(k)}=(n-m)x_j^{(k)}+\sum_{\substack{i=1\\i\neq k}}^{m}x_j^{(i)}\beta_{ik}$$

若能使得且对于所有的 j 有

$$n-m>\left|\sum_{\substack{i=1\\i\neq k}}^{m}x_j^{(i)}\beta_{ik}\right|$$

则 $x^{(k)}$ 是网络的吸引子。上式右端可进一步化为

$$\left|\sum_{\substack{i=1\\i\neq k}}^{m}x_j^{(i)}\beta_{ik}\right|\leqslant\sum_{\substack{i=1\\i\neq k}}^{m}\left|\beta_{ik}\right|\leqslant(m-1)\beta_m$$

式中，$\beta_m=\left|\beta_{ik}\right|_{\max}$，进而若能使得 $n-m>(m-1)\beta_m$，即

$$m<\frac{n+\beta_m}{1+\beta_m}$$

则可以保证所有的样本均为网络吸引子。

m 个样本满足

$$\alpha n\leqslant d_H(x^{(i)},x^{(j)})\leqslant(1-\alpha)n$$

式中，$i,j=1,2,\cdots,m,i\neq j,0<\alpha<0.5$，则有

$$\left|\beta_{ij}\right|\leqslant n-2\alpha n=\beta_m$$

从而得出 m 个样本均为网络吸引子的条件为

$$m<\frac{2n(1-\alpha)}{1+n(1-2\alpha)}$$

注意，上式仅为充分条件。当不满足上述条件时，需要具体检验才能确定。

4）记忆容量

所谓记忆容量是指在网络结构参数一定的条件下，要保证联想功能的正确实现，网络所能存储的最大的样本数。也就是说，给定网络节点数 n，样本数 m 最大可为多少，这些样本向量不仅本身应为网络的吸引子，而且应有一定的吸引域，这样才能实现联想记忆的功能。

记忆容量不仅与节点数 n 有关，它还与连接权的设计有关，适当地设计连接权可以提高网络的记忆容量。记忆容量还与样本本身的性质有关，对于用 Hebb 规则设计连接权的网

络，如果输入样本是正交的，则可以获得最大的记忆容量。实际问题的样本不可能都是正交的，故在研究记忆容量时通常都假设样本向量是随机的。

记忆容量还与要求的吸引域大小有关，要求的吸引域越大，记忆容量便越小。一个样本向量$\boldsymbol{x}^{(k)}$的吸引域可以看成是以该向量为中心的球体。在该球体中的向量$\boldsymbol{x}^{(s)}$满足

$$d_H(\boldsymbol{x}^{(s)},\boldsymbol{x}^{(k)})\leqslant\alpha n \qquad (0\leqslant\alpha\leqslant0.5)$$

式中，α为吸引半径。

对于给定的网络，严格的分析并确定其记忆容量并不是一件很容易的事情。Hopfield 曾提出了一个数量范围，即$m\leqslant0.15n$。按照样本为随机分布的假设的理论分析表明，当$n\to\infty$时，其记忆容量为

$$m<\frac{(1-2\alpha)^2 n}{2\ln n}$$

式中，α为要求的吸引半径。

上面提到，当样本为两两正交时，可以有最大的记忆容量。对于一般的记忆样本，可以通过改进连接权的设计来提高记忆容量。下面介绍其中的一种方法。

设给定m个样本向量$\boldsymbol{x}^{(k)}(k=1,2,\cdots,m)$，首先组成如下的$n\times(m-1)$阶矩阵

$$A=[x^{(1)}-x^{(m)},x^{(2)}-x^{(m)},\cdots,x^{(m-1)}-x^{(m)}]$$

对$\boldsymbol{A}$进行奇异值分解

$$\boldsymbol{A}=\boldsymbol{U\Sigma V}^{\mathrm{T}}$$

其中

$$\sum=\begin{bmatrix}S & 0\\ 0 & 0\end{bmatrix},\boldsymbol{S}=\mathrm{diag}(\sigma_1,\sigma_2,\cdots,\sigma_r)$$

式中，$\boldsymbol{U}$是$n\times n$正交阵；$\boldsymbol{V}$是$(m-1)\times(m-1)$正交阵；$\boldsymbol{U}$可表示成

$$\boldsymbol{U}=[\boldsymbol{u}_1,\boldsymbol{u}_2,\cdots,\boldsymbol{u}_r,\boldsymbol{u}_{r+1},\cdots,\boldsymbol{u}_n]$$

则$\boldsymbol{u}_1,\boldsymbol{u}_2,\cdots,\boldsymbol{u}_r$是对应于非零奇异值$\sigma_1,\sigma_2,\cdots,\sigma_r$的左奇异向量，且组成了$\boldsymbol{A}$的值域空间的正交基；$\boldsymbol{u}_{r+1},\cdots,\boldsymbol{u}_n$是$\boldsymbol{A}$的值域的正交补空间的正交基。

按如下方法组成连接权矩阵$\boldsymbol{W}$和阈值向量$\boldsymbol{\theta}$。

$$\boldsymbol{W}=\sum_{k=1}^{r}\boldsymbol{u}_k\boldsymbol{u}_k^{\mathrm{T}}$$

$$\boldsymbol{\theta}=\boldsymbol{W}\boldsymbol{x}^{(m)}-\boldsymbol{x}^{(m)}$$

显然，按上述方法求得的连接权矩阵是对称的，因而可以保证异步工作方式的稳定性。下面进一步证明给定的样本向量$\boldsymbol{x}^{(k)}(k=1,2,\cdots,m)$都是吸引子。

由于$\boldsymbol{u}_1,\boldsymbol{u}_2,\cdots,\boldsymbol{u}_r$是$\boldsymbol{A}$的值域空间的正交集，因此$\boldsymbol{A}$中的任一向量$\boldsymbol{x}^{(k)}-\boldsymbol{x}^{(m)}(k=1,2,\cdots,m-1)$均可表示为$\boldsymbol{u}_1,\boldsymbol{u}_2,\cdots,\boldsymbol{u}_r$的线性组合，即

$$\boldsymbol{x}^{(k)}-\boldsymbol{x}^{(m)}=\sum_{i=1}^{r}\alpha_i\boldsymbol{u}_i$$

由于 $\boldsymbol{U}$ 为正交阵，因为 $\boldsymbol{u}_1,\boldsymbol{u}_2,\cdots,\boldsymbol{u}_r$ 为互相正交的单位向量，从而对任一向量 $\boldsymbol{u}_i(i=1,2,\cdots,r)$ 有

$$\boldsymbol{W}\boldsymbol{u}_i=\sum_{k=1}^{r}\boldsymbol{u}_k\boldsymbol{u}_k^{\mathrm{T}}\boldsymbol{u}_i=\boldsymbol{u}_i$$

进而有

$$\boldsymbol{W}(\boldsymbol{x}^{(k)}-\boldsymbol{x}^{(m)})=\boldsymbol{W}\sum_{i=1}^{r}\alpha_i\boldsymbol{u}_i=\sum_{i=1}^{r}\alpha_i(\boldsymbol{W}\boldsymbol{u}_i)=\sum_{i=1}^{r}\alpha_i\boldsymbol{u}_i=\boldsymbol{x}^{(k)}-\boldsymbol{x}^{(m)}$$

对于任一样本向量 $\boldsymbol{x}^{(k)}(k=1,2,\cdots,m-1)$ 有

$$\boldsymbol{W}\boldsymbol{x}^{(k)}-\boldsymbol{\theta}=\boldsymbol{W}\boldsymbol{x}^{(k)}-\boldsymbol{W}\boldsymbol{x}^{(m)}+\boldsymbol{x}^{(m)}=\boldsymbol{W}(\boldsymbol{x}^{(k)}-\boldsymbol{x}^{(m)})+\boldsymbol{x}^{(m)}=\boldsymbol{x}^{(k)}$$

从而有

$$f(\boldsymbol{W}\boldsymbol{x}^{(k)}-\boldsymbol{\theta})=f(\boldsymbol{x}^{(k)})=\boldsymbol{x}^{(k)}$$

对于第 m 个样本 $\boldsymbol{x}^{(m)}$ 有

$$\boldsymbol{W}\boldsymbol{x}^{(m)}-\boldsymbol{\theta}=\boldsymbol{W}\boldsymbol{x}^{(m)}-\boldsymbol{W}\boldsymbol{x}^{(m)}+\boldsymbol{x}^{(m)}=\boldsymbol{x}^{(m)}$$

从而有

$$f(\boldsymbol{W}\boldsymbol{x}^{(m)}-\boldsymbol{\theta})=f(\boldsymbol{x}^{(m)})=\boldsymbol{x}^{(m)}$$

以上推证过程说明，按照这种方法设计的连接权矩阵，可以使得所有的样本 $\boldsymbol{x}^{(k)}(k=1,2,\cdots,m)$ 均为网络的吸引子，而不要求它们两两正交。也就是说，按此设计提高了网络的记忆容量。

例 1-7 对如图 1-37 所示的离散型 Hopfield 网络进行设计。其中网络节点数 $n=4$，$\theta_i=0$（$i=1,2,3,4$），样本数 $m=2$，两个样本为

$$\boldsymbol{x}^{(1)}=\begin{bmatrix}1\\1\\1\\1\end{bmatrix},\boldsymbol{x}^{(2)}=\begin{bmatrix}-1\\-1\\-1\\-1\end{bmatrix}$$

解：首先根据 Hebb 规则式（1-57）求得连接权矩阵为

$$\boldsymbol{W}=\boldsymbol{x}^{(1)}\boldsymbol{x}^{(1)T}+\boldsymbol{x}^{(2)}\boldsymbol{x}^{(2)T}-2\boldsymbol{I}=\begin{bmatrix}0&2&2&2\\2&0&2&2\\2&2&0&2\\2&2&2&0\end{bmatrix}$$

这里 $d_H(\boldsymbol{x}^{(1)},\boldsymbol{x}^{(2)})=4$，相等于 $\alpha=0$，显然它不满足上面给出的充分条件。$\boldsymbol{x}^{(1)}$ 和 $\boldsymbol{x}^{(2)}$ 是否是网络的吸引子需具体加以检验

$$f(\boldsymbol{W}\boldsymbol{x}^{(1)})=f\begin{bmatrix}6\\6\\6\\6\end{bmatrix}=\begin{bmatrix}1\\1\\1\\1\end{bmatrix}=\boldsymbol{x}^{(1)},f(\boldsymbol{W}\boldsymbol{x}^{(2)})=f\begin{bmatrix}-6\\-6\\-6\\-6\end{bmatrix}=\begin{bmatrix}-1\\-1\\-1\\-1\end{bmatrix}=\boldsymbol{x}^{(2)}$$

可见，两个样本 $\boldsymbol{x}^{(1)}$ 和 $\boldsymbol{x}^{(2)}$ 均为网络的吸引子。事实上，由于 $\boldsymbol{x}^{(2)}=-\boldsymbol{x}^{(1)}$，因此只要其中一个是吸引子，则另一个也必为吸引子。

下面再考察这两个吸引子是否具有一定的吸引能力，即是否具备联想记忆的功能。

（1）设 $\boldsymbol{x}(0)=\boldsymbol{x}^{(3)}=[-1\quad 1\quad 1\quad 1]^{\mathrm{T}}$，显然它比较接近 $\boldsymbol{x}^{(1)}$。下面用异步方式按 1,2,3,4 的调整次序来演变网络

$$x_1(1)=f\left(\sum_{j=1}^{4}w_{1j}x_j(0)\right)=f(6)=1$$

$$x_2(1)=x_2(0)=1\qquad x_3(1)=x_3(0)=1\qquad x_4(1)=x_4(0)=1$$

即 $\boldsymbol{x}(1)=[1\quad 1\quad 1\quad 1]^{\mathrm{T}}=\boldsymbol{x}^{(1)}$。可见，只需异步方式调整一步即可收敛到 $\boldsymbol{x}^{(1)}$。

（2）设 $\boldsymbol{x}(0)=\boldsymbol{x}^{(4)}=[1\quad -1\quad -1\quad -1]^{\mathrm{T}}$，显然它比较接近 $\boldsymbol{x}^{(2)}$。下面用异步方式按 1,2,3,4 的调整次序来演变网络

$$x_1(1)=f\left(\sum_{j=1}^{4}w_{1j}x_j(0)\right)=f(-6)=-1$$

$$x_2(1)=x_2(0)=-1\qquad x_3(1)=x_3(0)=-1\qquad x_4(1)=x_4(0)=-1$$

即 $\boldsymbol{x}(1)=[-1\quad -1\quad -1\quad -1]^{\mathrm{T}}=\boldsymbol{x}^{(2)}$。可见，只需异步方式调整一步即可收敛到 $\boldsymbol{x}^{(2)}$。

（3）设 $\boldsymbol{x}(0)=\boldsymbol{x}^{(5)}=[1\quad 1\quad -1\quad -1]^{\mathrm{T}}$，这时它与 $\boldsymbol{x}^{(1)}$ 和 $\boldsymbol{x}^{(2)}$ 的海明距离均为 2。若按 1,2,3,4 的调整次序调整网络可得

$$x_1(1)=f\left(\sum_{j=1}^{4}w_{1j}x_j(0)\right)=f(-2)=-1$$

$$x_i(1)=x_i(0)\qquad\qquad i=2,3,4$$

即 $\boldsymbol{x}(1)=[-1\quad 1\quad -1\quad -1]^{\mathrm{T}}$

$$x_2(2)=f\left(\sum_{j=1}^{4}w_{2j}x_j(1)\right)=f(-6)=-1$$

$$x_i(2)=x_i(1)\qquad\qquad i=1,3,4$$

即 $\boldsymbol{x}(2)=[-1\quad -1\quad -1\quad -1]^{\mathrm{T}}=\boldsymbol{x}^{(2)}$。可见此时 $\boldsymbol{x}^{(5)}$ 收敛到了 $\boldsymbol{x}^{(2)}$。

若按 3,4,1,2 的调整次序调整网络可得

$$x_3(1)=f\left(\sum_{j=1}^{4}w_{3j}x_j(0)\right)=f(2)=1$$

$$x_i(1)=x_i(0) \qquad i=1,2,4$$

即 $\boldsymbol{x}(1)=[1\quad 1\quad 1\quad -1]^{\mathrm{T}}$

$$x_4(2)=f\left(\sum_{j=1}^{4}w_{4j}x_j(1)\right)=f(6)=1$$

$$x_i(2)=x_i(1) \qquad i=1,2,3$$

即 $\boldsymbol{x}(2)=[1\quad 1\quad 1\quad 1]^{\mathrm{T}}=\boldsymbol{x}^{(1)}$。可见，此时 $\boldsymbol{x}^{(5)}$ 收敛到了 $\boldsymbol{x}^{(1)}$。

从上面具体计算可以看出，对于不同的调整次序，$\boldsymbol{x}^{(5)}$ 既可弱收敛到 $\boldsymbol{x}^{(1)}$，也可弱收敛到 $\boldsymbol{x}^{(2)}$。

下面对该例应用同步方式进行计算，仍取 $\boldsymbol{x}(0)$ 为 $\boldsymbol{x}^{(3)}$，$\boldsymbol{x}^{(4)}$ 和 $\boldsymbol{x}^{(5)}$ 三种情况。

（1）设 $\boldsymbol{x}(0)=\boldsymbol{x}^{(3)}=[-1\quad 1\quad 1\quad 1]^{\mathrm{T}}$，则

$$\boldsymbol{x}(1)=f[\boldsymbol{W}\boldsymbol{x}(0)]=f(\boldsymbol{W}\boldsymbol{x}^{(3)})=f\begin{bmatrix}6\\2\\2\\2\end{bmatrix}=\begin{bmatrix}1\\1\\1\\1\end{bmatrix},\boldsymbol{x}(2)=f[\boldsymbol{W}\boldsymbol{x}(1)]=f\begin{bmatrix}6\\6\\6\\6\end{bmatrix}=\begin{bmatrix}1\\1\\1\\1\end{bmatrix}$$

可见，此时 $\boldsymbol{x}^{(3)}$ 收敛到了 $\boldsymbol{x}^{(1)}$。

（2）设 $\boldsymbol{x}(0)=\boldsymbol{x}^{(4)}=[1\quad -1\quad -1\quad -1]^{\mathrm{T}}$，则

$$\boldsymbol{x}(1)=f[\boldsymbol{W}\boldsymbol{x}(0)]=f\begin{bmatrix}-6\\-2\\-2\\-2\end{bmatrix}=\begin{bmatrix}-1\\-1\\-1\\-1\end{bmatrix},\boldsymbol{x}(2)=f[\boldsymbol{W}\boldsymbol{x}(1)]=f\begin{bmatrix}-6\\-6\\-6\\-6\end{bmatrix}=\begin{bmatrix}-1\\-1\\-1\\-1\end{bmatrix}$$

可见，此时 $\boldsymbol{x}^{(4)}$ 收敛到了 $\boldsymbol{x}^{(2)}$。

（3）设 $\boldsymbol{x}(0)=\boldsymbol{x}^{(5)}=[1\quad 1\quad -1\quad -1]^{\mathrm{T}}$，则

$$\boldsymbol{x}(1)=f[\boldsymbol{W}\boldsymbol{x}(0)]=f\begin{bmatrix}-2\\-2\\2\\2\end{bmatrix}=\begin{bmatrix}-1\\-1\\1\\1\end{bmatrix},\boldsymbol{x}(2)=f[\boldsymbol{W}\boldsymbol{x}(1)]=f\begin{bmatrix}2\\2\\-2\\-2\end{bmatrix}=\begin{bmatrix}1\\1\\-1\\-1\end{bmatrix}=\boldsymbol{x}(0)$$

可见，它将在两个状态间跳跃，产生极限环为 2 的自持振荡。若根据前面的稳定性分析，由于此时连接权矩阵 $\boldsymbol{W}$ 不是非负定阵，因此出现了振荡。

因为网络有四个节点，所以有 2^4=16 个状态（阈值取 0），其中只有以上两个状态 $\boldsymbol{x}^{(1)}$ 和 $\boldsymbol{x}^{(2)}$ 是稳定的，原因是它们满足式（1-57），其余状态都会收敛到与之邻近的稳定状态

上，所以说这种网络具有一定的纠错能力。

例 1-8 对如图 1-37 所示的离散型 Hopfield 网络进行设计。其中网络节点数 $n=4$，$\theta_i=0$（$i=1,2,3,4$），样本数 $m=2$，两个样本为

$$x^{(1)}=\begin{bmatrix}1\\0\\1\\0\end{bmatrix},x^{(2)}=\begin{bmatrix}0\\1\\0\\1\end{bmatrix}$$

解：首先根据 Hebb 规则式（1-58）求得连接权矩阵为

$$w_{12}=(2x_1^{(1)}-1)(2x_2^{(1)}-1)+(2x_1^{(2)}-1)(2x_2^{(2)}-1)=(1)\times(-1)+(-1)\times(1)=-2=w_{21}$$
$$w_{13}=(2x_1^{(1)}-1)(2x_3^{(1)}-1)+(2x_1^{(2)}-1)(2x_3^{(2)}-1)=(1)\times(1)+(-1)\times(-1)=2=w_{31}$$
$$w_{14}=(2x_1^{(1)}-1)(2x_4^{(1)}-1)+(2x_1^{(2)}-1)(2x_4^{(2)}-1)=(1)\times(-1)+(-1)\times(1)=-2=w_{41}$$
$$w_{23}=(2x_2^{(1)}-1)(2x_3^{(1)}-1)+(2x_2^{(2)}-1)(2x_3^{(2)}-1)=(-1)\times(1)+(1)\times(-1)=-2=w_{32}$$
$$w_{24}=(2x_2^{(1)}-1)(2x_4^{(1)}-1)+(2x_2^{(2)}-1)(2x_4^{(2)}-1)=(-1)\times(-1)+(1)\times(1)=2=w_{42}$$
$$w_{34}=(2x_3^{(1)}-1)(2x_4^{(1)}-1)+(2x_3^{(2)}-1)(2x_4^{(2)}-1)=(1)\times(-1)+(-1)\times(1)=-2=w_{43}$$

则网络的连接权矩阵为

$$W=\begin{bmatrix}0&-2&2&-2\\-2&0&-2&2\\2&-2&0&-2\\-2&2&-2&0\end{bmatrix}$$

同理可以证明，在网络的 $2^4=16$ 个状态（阈值取 0）中，只有以上两个状态 $x^{(1)}$ 和 $x^{(2)}$ 是稳定的，原因是它们满足式（1-58），其余状态都会收敛到与之邻近的稳定状态上。

2. 连续型 Hopfield 网络

1）网络的结构和工作方式

连续型神经网络的各神经元是并行（同步）工作的。它也是单层的反馈网络，结构仍如图 1-37 所示。对于每一个神经元节点，其工作方式为

$$\begin{cases}s_i=\sum_{j=1}^{n}w_{ij}x_j-\theta_j\\ \dfrac{dy_i}{dt}=-\dfrac{1}{\tau}y_i+s_i\\ x_i=f(y_i)\end{cases}\tag{1-60}$$

这里，同样假定 $w_{ij}=w_{ji}$，它与离散型 Hopfield 网络相比，这里多了中间一个式子，该式是一阶微分方程，相当于一阶惯性环节。s_i 是该环节的输入，y_i 是该环节的输出。对于离

散型 Hopfield 网络，中间的式子也可看成为 $y_i = s_i$。它们之间的另一个差别是第三个式子一般不再是二值函数，而一般取 S 型函数，即当 $x_i \in (-1,1)$ 时，取

$$x_i = f(y_i) = \frac{1-\mathrm{e}^{-\mu y_i}}{1+\mathrm{e}^{-\mu y_i}}$$

当 $x_i \in (0,1)$ 时，取

$$x_i = f(y_i) = \frac{1}{1+\mathrm{e}^{-\mu y_i}}$$

它们都是连续的单调上升的函数，如图 1-38 所示。

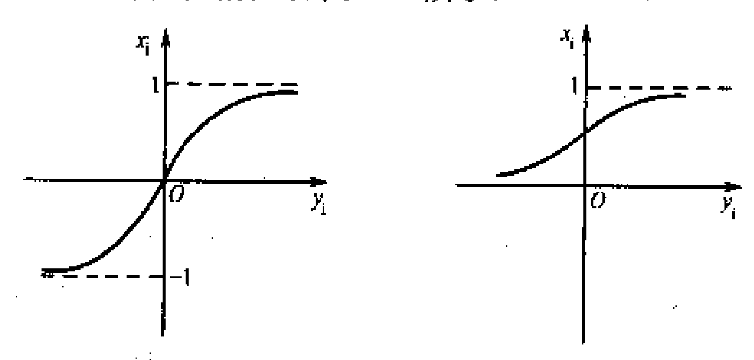


图 1-38　变换函数离散型 Hopfield 网络

Hopfield 利用模拟电路设计了一个连续型 Hopfield 网络的电路模型。图 1-39 所示为其中由运算放大器电路实现的一个节点的模型。

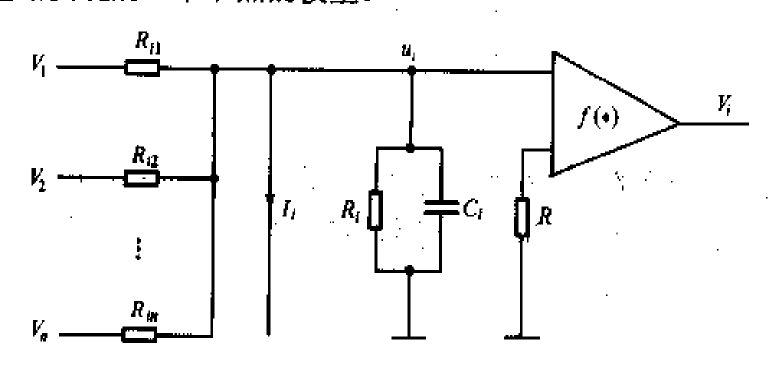


图 1-39　连续型 Hopfield 网络的电路模型

根据图 1-39，可以列出如下的电路方程，即

$$\begin{cases} C_i \dfrac{\mathrm{d}u_i}{\mathrm{d}t} + \dfrac{u_i}{R_i} + I_i = \displaystyle\sum_{j=1}^{n} \dfrac{V_j - u_i}{R_{ij}} \\ V_i = f(u_i) \end{cases}$$

经整理得

$$\begin{cases} \dfrac{\mathrm{d}u_i}{\mathrm{d}t} = -\dfrac{1}{R_i' C_i} u_i + \displaystyle\sum_{j=1}^{n} \dfrac{1}{R_{ij} C_i} V_j - \dfrac{I_i}{C_i} \\ V_i = f(u_i) \end{cases}$$

其中 $\dfrac{1}{R_i'}=\dfrac{1}{R_i}+\sum\limits_{j=1}^{n}\dfrac{1}{R_{ij}}$

若令 $x_i=V_i$， $y_i=u_i$， $\tau=R_i'C_i$， $w_{ij}=\dfrac{1}{R_{ij}C_i}$， $\theta_i=\dfrac{I_i}{C_i}$

则上式化为

$$\begin{cases}\dfrac{\mathrm{d}y_i}{\mathrm{d}t}=-\dfrac{1}{\tau}y_i+\sum\limits_{j=1}^{n}w_{ij}x_j-\theta_i\\ x_i=f(y_i)\end{cases}\tag{1-61}$$

式中，$f(\bullet)$为常用 Sigmoid 函数，即

$$x_i=f(y_i)=\frac{1}{2}\left[1+\tanh\left(\frac{y_i}{y_0}\right)\right]$$

可以看出，连续型 Hopfield 网络实质上是一个连续的非线性动力学系统。它可用一组非线性微分方程来描述。当给定初始状态 $x_i(0)(i=1,2,\cdots,n)$ 时，通过求解非线性微分方程组可求得网络状态的运动轨迹。若系统是稳定的，则它最终可收敛到一个稳定状态。若用如图 1-40 所示的硬件来实现，则这个求解非线性微分方程的过程将由该电路自动完成，其求解速度是非常快的。

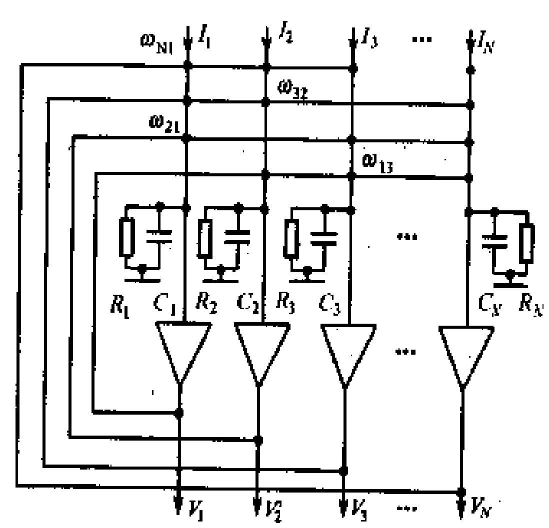


图 1-40　用运算放大器构造的连续型 Hopfield 网络

2）稳定性

定义连续 Hopfield 网络能量函数为

$$\begin{aligned}E&=-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}w_{ij}x_jx_i+\sum_{i=1}^{n}x_i\theta_i+\sum_{i=1}^{n}\frac{1}{\tau_i}\int_0^{x_i}f^{-1}(\eta)\mathrm{d}\eta\\&=-\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x}+\boldsymbol{x}^{\mathrm{T}}\boldsymbol{\theta}+\sum_{i=1}^{n}\frac{1}{\tau_i}\int_0^{x_i}f^{-1}(\eta)\mathrm{d}\eta\end{aligned}\tag{1-62}$$

该能量函数的表达式与离散型Hopfield网络的定义是完全相同的。对于离散型Hopfield网络，由于$f(\bullet)$是二值函数，因此第三项的积分项为零。由于$x_i \in (-1,1)$或$x_i \in (0,1)$，因此上述定义的能量函数E是有界的，只需证明$\mathrm{d}E/\mathrm{d}t \leqslant 0$，即可说明系统是稳定的，因

$$\frac{\mathrm{d}E}{\mathrm{d}t}=\sum_{i=1}^{n}\frac{\partial E}{\partial x_i}\frac{\mathrm{d}x_i}{\mathrm{d}t}$$

根据式（1-62）所述E的表达式可以求得

$$\frac{\partial E}{\partial x_i}=-\sum_{j=1}^{n}w_{ij}x_j+\theta_i+\frac{1}{\tau_i}f^{-1}(x_i)=-\sum_{j=1}^{n}w_{ij}x_j+\theta_i+\frac{1}{\tau_i}y_i=-\frac{\mathrm{d}y_i}{\mathrm{d}t} \tag{1-63}$$

代入上式得
$$\frac{\mathrm{d}E}{\mathrm{d}t}=\sum_{i=1}^{n}\left(-\frac{\mathrm{d}y_i}{\mathrm{d}t}\frac{\mathrm{d}x_i}{\mathrm{d}t}\right)=-\sum_{i=1}^{n}\left(\frac{\mathrm{d}y_i}{\mathrm{d}x_i}\frac{\mathrm{d}x_i}{\mathrm{d}t}\frac{\mathrm{d}x_i}{\mathrm{d}t}\right)=-\sum_{i=1}^{n}\left(\frac{\mathrm{d}y_i}{\mathrm{d}x_i}\left(\frac{\mathrm{d}x_i}{\mathrm{d}t}\right)^2\right)$$

前面已假设$x_i=f(y_i)$是单调上升函数，如图1-38所示。显然它的反函数$y_i=f^{-1}(x_i)$为单调上升函数，即有$\mathrm{d}y_i/\mathrm{d}x_i>0$。同时，$(\mathrm{d}x_i/\mathrm{d}t)^2 \geqslant 0$，因而有

$$\frac{\mathrm{d}E}{\mathrm{d}t}\leqslant 0 \quad \text{（所有}x_i\text{均为常数时才取等号）} \tag{1-64}$$

根据李雅普诺夫稳定性理论，该网络系统一定是渐近稳定的，即随着时间的演变，网络状态总是朝E减小的方向运动，一直到E取得极小值，这时所有的x_i变为常数，即网络收敛到稳定状态。

在应用连续型Hopfield网络解决实际问题时，如果能将某个待研究解决的问题，化为一个计算能量函数，且使这个能量函数的最小值正好对应于一定约束条件下问题的解答时，则此问题就可以用连续型Hopfield网络来求解了。连续型Hopfield网络主要用来进行优化计算。因此，如何设计连接权系数及其他参数，需根据具体问题来加以确定。下面以连续型Hopfleld神经网络应用于TSP（Travelling Salesman Problem）为例加以说明。TSP问题是人工智能中的一个难题。

例1-9 推销员要到n个城市去推销产品，要求推销员每个城市都要去到，且只能去一次，如何规划路线才能使所走的路程最短。利用连续型Hopfield网络来进行优化计算。

解：这是一个典型的组合优化问题。下面要解决的问题是如何恰当地描述该问题，使其适合于用Hopfield网络来求解。正是由于Hopfield成功地求解了TSP问题，才使得人们对神经网络再次引起了广泛的兴趣。

设使用n^2个神经元节点组成如下的方阵排列（以n=5为例），见表1-2。

表1-2 神经元节点方阵排列

	1	2	3	4	5
A	0	1	0	0	0
B	0	0	0	1	0
C	1	0	0	0	0
D	0	0	0	0	1
E	0	0	1	0	0

每个神经元采用如下的 S 型变换函数

$$x_{\alpha i}=\frac{1}{1+\mathrm{e}^{-\mu s_{\alpha i}}}$$

式中，$\alpha\in\{A,B,C,D,E\},i\in\{1,2,3,4,5\}$。这里取较大的$\mu$，以使 S 型函数比较陡峭，从而稳态时$x_{\alpha i}$能够趋于 1 或趋于 0。

在表 1-2 所列的方阵中，A、B、C、D、E 表示城市名称，1、2、3、4、5 表示路径顺序。为了保证每个城市只去一次，方阵中每行只能有一个元素为 1，其余为零。为了在某一时刻只能经一个城市，方阵中每列也只能有一个元素为 1，其余为零。为使每个城市必须经一次，方阵中 1 的个数总和必须为 n。对于所给方阵，其相应的路径顺序为：$\mathrm{C}\to\mathrm{A}\to\mathrm{E}\to\mathrm{B}\to\mathrm{D}$，所走路程的总距离为

$$d=d_{\mathrm{CA}}+d_{\mathrm{AE}}+d_{\mathrm{EB}}+d_{\mathrm{BD}}=\frac{\rho_1}{2}\sum_{\alpha}\sum_{\beta\neq\alpha}\sum_{i}d_{\alpha\beta}x_{\alpha i}x_{\beta,i+1}+\frac{\rho_1}{2}\sum_{\alpha}\sum_{\beta\neq\alpha}\sum_{i}d_{\alpha\beta}x_{\alpha i}x_{\beta,i-1}$$

式中，$d_{\alpha\beta}$表示城市α 到城市β 的距离；$x_{\alpha i}$表示矩阵中的第α 行第 i 列的元素，其值为 1 时，表示第 i 步访问城市α，为 0 时，表示第 i 步不访问城市α；$\sum\limits_{\alpha}x_{\alpha i}=1$表示对于所有的 i 每个城市必须访问一次；$\sum\limits_{i}x_{\alpha i}=1$表示对于所有的$\alpha$ 每个城市只能访问一次。

根据路径最短的要求及上述约束条件可以写出总的能量函数为

$$E=\frac{\rho_1}{2}\sum_{\alpha}\sum_{\beta\neq\alpha}\sum_{i}(d_{\alpha\beta}x_{\alpha i}x_{\beta,i+1}+d_{\alpha\beta}x_{\alpha i}x_{\beta,i-1})+\frac{\rho_2}{2}\sum_{i}\sum_{\alpha}\sum_{\beta\neq\alpha}x_{\alpha i}x_{\beta i}+$$

$$\frac{\rho_3}{2}\sum_{\alpha}\sum_{i}\sum_{j\neq i}x_{\alpha i}x_{\alpha j}+\frac{\rho_4}{2}\left(\sum_{\alpha}\sum_{i}x_{\alpha i}-n\right)^2+\sum_{\alpha}\sum_{i}\frac{1}{\tau_{\alpha i}}\int_0^{x_{\alpha i}}f^{-1}(\eta)\mathrm{d}\eta$$

式中，第一项反映了路径的总长度，如当$\alpha=\mathrm{C},\beta=\mathrm{A},i=1$，且神经网络状态如上面方阵排列时，有$d_{\alpha\beta}x_{\alpha i}x_{\beta,i+1}=d_{\mathrm{CA}}x_{\mathrm{C1}}x_{\mathrm{A2}}$，注意此处若当 $i+1>n$ 时，用 1 代 $i+1$；第二项反映了“方阵的每一列只能有一个元素为 1”的要求，即每一列只能有一个元素为 1 时 E 最小；第三项反映了“方阵的每一行只能有一个元素为 1”的要求，即每一行只能有一个元素为 1 时 E 最小；第四项反映了“方阵中 1 的个数总和为 n”的要求，即方阵中 1 的个数总和为 n 时 E 最小；第五项是 Hopfield 网络本身的要求；$\rho_1,\rho_2,\rho_3,\rho_4$是各项的加权系数。

由上式可以求得

$$\frac{\partial E}{\partial x_{\alpha i}}=\rho_1\sum_{\beta\neq\alpha}d_{\alpha\beta}(x_{\beta,i+1}+x_{\beta,i-1})+\rho_2\sum_{\beta\neq\alpha}x_{\beta i}+\rho_3\sum_{j\neq i}x_{\alpha j}+\rho_4(\sum_{\alpha}\sum_{i}x_{\alpha i}-n)+$$

$$\frac{1}{\tau_{\alpha i}}f^{-1}(x_{\alpha i})$$

根据前面关于稳定性的分析式（1-63），应有如下关系成立，即

$$\frac{\mathrm{d}y_{\alpha i}}{\mathrm{d}t}=-\frac{\partial E}{\partial x_{\alpha i}}$$

代入上式可得网络的动态方程为

$$\frac{\mathrm{d}y_{\alpha i}}{\mathrm{d}t}=-\frac{1}{\tau_{\alpha i}}y_{\alpha i}-\rho_1\sum_{\beta\neq\alpha}d_{\alpha\beta}(x_{\beta,i+1}+x_{\beta,i-1})-\rho_2\sum_{\beta\neq\alpha}x_{\beta i}-\rho_3\sum_{j\neq i}x_{\alpha j}-\rho_4\left(\sum_{\alpha}\sum_{i}x_{\alpha i}-n\right)$$

令

$$\begin{cases}w_{\alpha i,\beta j}=-\rho_1 d_{\alpha\beta}(\delta_{j,i+1}+\delta_{j,i-1})-\rho_2\delta_{ij}(1-\delta_{\alpha\beta})-\rho_3\delta_{\alpha\beta}(1-\delta_{ij})-\rho_4\\ \theta_i=-\rho_4 n\end{cases}$$

式中，$\delta_{\alpha\beta},\delta_{ij}$ 是离散 δ 函数，即

$$\delta_{ij}=\begin{cases}1 & i=j\\ 0 & i\neq j\end{cases}$$

从而可将以上问题表示成以下连续的 Hopfield 网络格式

$$\begin{cases}\dfrac{\mathrm{d}y_{\alpha i}}{\mathrm{d}t}=-\dfrac{1}{\tau_{\alpha i}}y_{\alpha i}+\sum\limits_{\beta}\sum\limits_{j}w_{\alpha i,\beta j}x_{\beta j}-\theta_i\\ x_{\alpha i}=f(y_{\alpha i})=\dfrac{1}{2}\left[1+\tanh\left(\dfrac{y_{\alpha i}}{y_0}\right)\right]\end{cases}$$

选择适当的参数值ρ_1, ρ_2, ρ_3, ρ_4和初值 y_0，按上式迭代直到收敛，其稳态解即为所要求的解，或根据以上方程构成连续的 Hopfield 网络，并适当地给定 $x_{\alpha i}(0)$（若无先验知识，可随机给定），运行该神经网络，其稳态解即为所要求的解。

例 1-10 聚类问题。

对于 N 个模式（可看做 N 维空间中的 N 个点）要聚成 K 类，使各类本身内的点最近。一般地说，可能的划分方法数为 $K^N/K!$。若用穷举法，则工作量是指数型的。划分的准则最常用的是平方误差。用 N 维向量表示各模式：$\{r_i:i=1,2,\cdots,N\}$，最优划分应使 $X^2=\sum_{i=1}^{n}(r_i^{(p)}-R_p)$ 最小，R_p 为 p 类的中心，即 $R_p=\sum_{i=1}^{N_p}r_i^{(p)}$（$p$=1,2,…,$K$; N_p 为第 p 类中的点数）。如对于一个 N=10, K=3 的聚类问题，可表示成如表 1-3 所列的矩阵形式。

表 1-3 聚类分布情况

	1	2	3	4	5	6	7	8	9	10
C_1	1	1	0	0	0	1	0	0	1	0
C_2	0	0	0	1	1	0	0	0	0	0
C_3	0	0	1	0	0	0	1	1	0	1

解：容易看出第 1，2，6，9 四个点属于 C_1 类，第 4，5 两个点属于 C_2 类，第 3，7，8，10 四个点属于 C_3 类。能量函数可写为

$$E=\frac{a}{2}\sum_{i=1}^{N}\sum_{p=1}^{K}\sum_{q\neq p}^{K}v_{pi}v_{qi}+\frac{b}{2}\sum_{i=1}^{N}(\sum_{p=1}^{K}v_{pi}-1)^2+\frac{c}{2}\sum_{p=1}^{K}\sum_{i=1}^{N}R_{pi}v_{pi}^2$$

式中，$p,q(p,q=1,2,\cdots,K)$为聚类号。第一项约束为每列（表示每个点）只能有一个 1；第二项约束为每列必须有一个 1，这两个约束表示每个点必须属于一类且只属于一类；第三项是使类内总距离最小。这里

$$R_{pi}=(x_i-x_p)^2+(y_i-y_p)^2$$

式中，x_i, y_i 是各点坐标（以二维为例，N=2）。x_p, y_p 是类中心的坐标，即

$$x_p=\sum_{i=1}^{N}x_i v_{pi}\bigg/\sum_{i=1}^{N}v_{pi}\,,\,y_p=\sum_{i=1}^{N}y_i v_{pi}\bigg/\sum_{i=1}^{N}v_{pi}$$

1.2.10 Boltzmann 神经网络

Boltzmann 机是一种随机神经网络，也是一种反馈型神经网络，它在很多方面与离散型 Hopfield 网络类似。Boltzmann 机可用于模式分类、预测、组合优化及规划等方面。

1. 网络的结构和工作方式

在结构上，Boltzmann 机是单层的反馈网络，形式上与离散型 Hopfield 网络一样，具有对称的连接权系数，即 $w_{ij}=w_{ji}$ 且 $w_{ii}=0$。但在功能上，Boltzmann 可以看成是多层网络，其中一部分节点是输入节点，一部分是输出节点，还有一部分是隐节点。隐节点不与外界发生联系，它主要用来实现输入与输出之间的高阶联系。

Boltzmann 机是一个随机神经网络，这是与 Hopfield 网络的最大区别。图 1-41 为其中一个神经元节点的示意图。

它的工作方式为

$$\begin{cases}s_i=\sum_{i=1}^{n}w_{ij}x_j-\theta_i\\ p_i=\dfrac{1}{1+e^{-s_i/T}}\end{cases}\tag{1-65}$$

式中，p_i 是 $x_i=1$ 的概率。显然，$x_i=0$（或 $x_i=-1$）的概率为

$$1-p_i=\frac{e^{-s_i/T}}{1+e^{-s_i/T}}\tag{1-66}$$

显然，p_i 是 S 型函数。T 越大，曲线越平坦；T 越小，曲线越陡峭。参数 T 通常称为"温区"，图 1-42 表示 S 型曲线随参数 T 变化的情况。当 $T\to 0$ 时，S 型函数便趋于二值函数，随机神经网络便退化为确定性网络，即与 Hopfield 网络具有同样的工作方式。

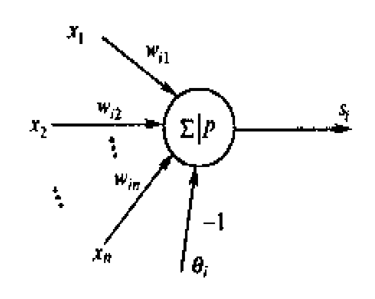


图 1-41 Boltzmann 机的单个神经元

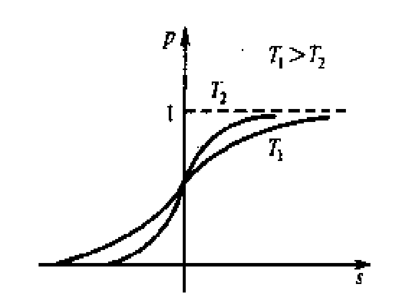


图 1-42 S 型曲线随温度的变化

与 Hopfield 网络一样，对 Boltzmann 机也定义网络的能量函数为

$$E=-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}w_{ij}x_jx_i+\sum_{i=1}^{n}x_i\theta_i=-\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x}+\boldsymbol{x}^{\mathrm{T}}\boldsymbol{\theta} \tag{1-67}$$

Boltzmann 机也有两种类型的工作方式：同步方式和异步方式。下面只考虑异步工作。若考虑第 i 个神经元的状态发生变化，根据前面讨论 Hopfield 网络时所作的推导，有

$$\Delta E_i=-\Delta x_i(\sum_{j=1}^{n}w_{ij}x_j-\theta_i)=-\Delta x_i s_i \tag{1-68}$$

这时 $x_i=1$ 的概率

$$p_i=\frac{1}{1+\mathrm{e}^{-s_i/T}} \tag{1-69}$$

若 $s_i>0$，则 $p_i>0.5$，即有较大的概率取 $x_i=1$。若原来 $x_i=1$；则 $\Delta x_i=0,\Delta E_i=0$；若原来 $x_i=0$，则 $\Delta x_i>0$，而此时 $s_i>0$，故 $\Delta E_i<0$。

如若 $s_i<0$，则有较大的概率取 $x_i=0$；若原来 $x_i=0$，则 $\Delta E_i=0$；若原来 $x_i=1$，则 $\Delta x_i<0$，而此时 $s_i<0$，故此时 $\Delta E_i<0$。

在以上任何情况下，随着系统状态的演变，从概率的意义上，系统的能量总是朝小的方向变化，系统最后总能稳定到能量的极小点附近。由于这是随机网络，因此在能量的极小点附近，系统也会停止在某一个固定的状态。

由于神经元状态按概率取值，因此以上分析只是从概率意义上说网络的能量总的趋势是朝减小的方向演变，但在有些步神经元状态可能按小概率取值，从而使能量增加，在有些情况下，这对跳出局部极值是有好处的。这也是 Boltzmann 网络和 Hopfield 网络的另一个不同之处。

为了有效地演化到网络能量函数的全局极小点，通常采用模拟退火的方法。开始采用较高的温度 T，此时各状态出现概率的差异不大，比较容易跳出局部极小点进入全局极小点附近，然后再逐渐降低温度 T，各状态出现概率的差别逐渐拉大，从而一方面可较准确地运动到能量的极小点，同时阻止它跳出该最小点。

根据前面的结果，当 x_i 由 1 变为 0 时，$\Delta x_i=-1$

则

$$\Delta E_i = E\big|_{x_i=0} - E\big|_{x_i=1} = \sum_{j=1}^{n} w_{ij}x_j - \theta_i = s_i \tag{1-70}$$

设 $x_i = 1$（其他状态不变）的概率为 p_1，相应的能量函数为 E_1， $x_i = 0$（其他状态不变）的概率为 p_0，相应的能量函数为 E_0，则有

$$p_1 = \frac{1}{1+\mathrm{e}^{-s_i/T}} = \frac{1}{1+\mathrm{e}^{-\Delta E_i/T}} \tag{1-71}$$

$$p_0 = 1 - p_1 = \frac{\mathrm{e}^{-\Delta E_i/T}}{1+\mathrm{e}^{-\Delta E_i/T}} \tag{1-72}$$

显然有

$$\frac{p_0}{p_1} = \mathrm{e}^{-\Delta E_i/T} = \mathrm{e}^{-(E_0-E_1)/T} = \frac{\mathrm{e}^{-E_0/T}}{\mathrm{e}^{-E_1/T}} \tag{1-73}$$

推而广之，容易得到，对于网络中任意两个状态 α 和 β 出现的概率与它们的能量 E_α 和 E_β 之间也满足

$$\frac{p_\alpha}{p_\beta} = \mathrm{e}^{-(E_\alpha-E_\beta)/T} = \frac{\mathrm{e}^{-E_\alpha/T}}{\mathrm{e}^{-E_\beta/T}} \tag{1-74}$$

这正好是 Boltzmann 分布，也就是该网络称为 Boltzmann 机的由来。

从以上结果可以看出，Boltzmann 机处于某一状态的概率主要取决于在此状态下的能量，能量越低，概率越大；同时，此概率还取决于温度参数 T，T 越大，不同状态出现概率的差异便越小，较容易跳出能量的局部极小点；T 越小时情形正好相反，这也就是采取模拟退火方法寻求全局最优的原因所在。

Boltzmann 机的实际运行也分为两个阶段：第一阶段是学习和训练阶段，即根据学习样本对网络进行训练，将知识分布地存储于网络的连接权中；第二阶段是工作阶段，即根据输入运行网络得到合适的输出，这一步实质上是按照某种机制将知识提取出来。

2．网络的学习和训练

网络学习的目的是通过给出一组学习样本，经学习后得到 Boltzmann 机各个神经元之间的连接权 w_{ij}。

设 Boltzmann 机共有 n 个节点，其中 m 个是输入和输出节点，其余 $r=n-m$ 个是隐节点。设 $\boldsymbol{x}_\alpha$表示 m 维输入和输出节点向量，$\boldsymbol{y}_\beta$表示 r 维隐节点向量，$\boldsymbol{x}_\alpha \wedge \boldsymbol{y}_\beta$表示整个网络的状态向量。同时设 $p^+(\boldsymbol{x}_\alpha)$表示学习样本出现的概率，即输入输出节点约束为 $\boldsymbol{x}_\alpha$时的概率。$\boldsymbol{x}_\alpha$最多可为 2^m 个，所以 $p^+(\boldsymbol{x}_\alpha)$也为 2^m 个。当学习样本数少于 2^m 个时，可取其余未给定样本的概率为 0。设 $p^+(\boldsymbol{x}_\alpha)$均为已知，且有

$$\sum_{\alpha=1}^{2^m} p^+(\boldsymbol{x}_\alpha)=1 \tag{1-75}$$

设 $p^-(\boldsymbol{x}_\alpha)$表示系统无约束时出现 $\boldsymbol{x}_\alpha$的概率，对网络进行学习的目的便是调整连接权，以使得 $p^-(\boldsymbol{x}_\alpha)$尽可能与 $p^+(\boldsymbol{x}_\alpha)$相一致。定义

$$G=\sum_{\alpha=1}^{2^m} p^+(\boldsymbol{x}_\alpha)\ln\left(\frac{p^+(\boldsymbol{x}_\alpha)}{p^-(\boldsymbol{x}_\alpha)}\right) \tag{1-76}$$

为使 $p^-(\boldsymbol{x}_\alpha)$与 $p^+(\boldsymbol{x}_\alpha)$相一致的度量，$G$ 是一个非负量，仅当 $p^+(\boldsymbol{x}_\alpha)=p^-(\boldsymbol{x}_\alpha)$时才有 $G=0$。因此下面的问题变为寻求连接权系数，以使 G 取极小值。根据上式可以求得

$$\frac{\partial G}{\partial w_{ij}}=-\sum_{\alpha=1}^{2^m}\frac{p^+(\boldsymbol{x}_\alpha)}{p^-(\boldsymbol{x}_\alpha)}\frac{\partial p^-(\boldsymbol{x}_\alpha)}{\partial w_{ij}} \tag{1-77}$$

根据上述假定有

$$p^-(\boldsymbol{x}_\alpha)=-\sum_{\beta=1}^{2^r} p^-(\boldsymbol{x}_\alpha\wedge\boldsymbol{y}_\beta)=\frac{\sum\limits_{\beta=1}^{2^r}\mathrm{e}^{-E_{\alpha\beta}/T}}{\sum\limits_{\lambda=1}^{2^m}\sum\limits_{\mu=1}^{2^r}\mathrm{e}^{-E_{\lambda\mu}/T}} \tag{1-78}$$

式中，$E_{\alpha\beta}$表示网络状态为 $\boldsymbol{x}_\alpha\wedge\boldsymbol{y}_\beta$时的能量，$E_{\lambda\mu}$表示网络状态为 $\boldsymbol{x}_\lambda\wedge\boldsymbol{y}_\mu$的能量，即（设神经元阈值$\theta_i=0$）

$$E_{\alpha\beta}=-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_i^{\alpha\beta}x_j^{\alpha\beta} \tag{1-79}$$

$$E_{\lambda\mu}=-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_i^{\lambda\mu}x_j^{\lambda\mu} \tag{1-80}$$

式中，$x_i^{\alpha\beta}$表示 n 维状态向量 $\boldsymbol{x}_\alpha\wedge\boldsymbol{y}_\beta$的第 i 个分量；$x_i^{\lambda\mu}$表示 n 维状态向量 $\boldsymbol{x}_\lambda\wedge\boldsymbol{y}_\mu$的第 i 个分量，进而求得

$$\begin{aligned}\frac{\partial p^-(\boldsymbol{x}_\alpha)}{\partial w_{ij}}=&\left[-\frac{1}{T}\sum_{\beta=1}^{2^r}\mathrm{e}^{-E_{\alpha\beta}/T}(-x_i^{\alpha\beta}x_j^{\alpha\beta})\sum_{\lambda=1}^{2^m}\sum_{\mu=1}^{2^r}\mathrm{e}^{-E_{\lambda\mu}/T}+\right.\\&\left.\frac{1}{T}\sum_{\lambda=1}^{2^m}\sum_{\mu=1}^{2^r}\mathrm{e}^{-E_{\lambda\mu}/T}(-x_i^{\lambda\mu}x_j^{\lambda\mu})\sum_{\beta=1}^{2^r}e^{-E_{\alpha\beta}/T}\right]\Big/(\sum_{\lambda=1}^{2^m}\sum_{\mu=1}^{2^r}\mathrm{e}^{-E_{\lambda\mu}/T})^2\\=&\frac{1}{T}[\sum_{\beta=1}^{2^r}p^-(\boldsymbol{x}_\alpha\wedge\boldsymbol{y}_\beta)x_i^{\alpha\beta}x_j^{\alpha\beta}-p^-(\boldsymbol{x}_\alpha)\sum_{\lambda=1}^{2^m}\sum_{\mu=1}^{2^r}p^-(\boldsymbol{x}_\lambda\wedge\boldsymbol{y}_\mu)x_i^{\lambda\mu}x_j^{\lambda\mu}]\end{aligned} \tag{1-81}$$

将上式代入前面$\partial G/\partial w_{ij}$的表达式中得

$$\frac{\partial G}{\partial w_{ij}}=-\frac{1}{T}[\sum_{\alpha=1}^{2^m}\sum_{\beta=1}^{2^r}\frac{p^+(\boldsymbol{x}_\alpha)}{p^-(\boldsymbol{x}_\alpha)}p^-(\boldsymbol{x}_\alpha\wedge\boldsymbol{y}_\beta)x_i^{\alpha\beta}x_j^{\alpha\beta}-(\sum_{\alpha=1}^{2^m}p^+(\boldsymbol{x}_\alpha))\sum_{\lambda=1}^{2^m}\sum_{\mu=1}^{2^r}p^-(\boldsymbol{x}_\lambda\wedge\boldsymbol{y}_\mu)x_i^{\lambda\mu}x_j^{\lambda\mu}]$$

因为

$$
\begin{aligned}
&p^{+}(x_{\alpha}\wedge y_{\beta})=p^{+}(y_{\beta}|x_{\alpha})p^{+}(x_{\alpha})\\
&p^{-}(x_{\alpha}\wedge y_{\beta})=p^{-}(y_{\beta}|x_{\alpha})p^{-}(x_{\alpha})\\
&p^{+}(y_{\beta}|x_{\alpha})=p^{-}(y_{\beta}|x_{\alpha})
\end{aligned}
\tag{1-82}
$$

所以

$$
\frac{p^{+}(x_{\alpha})}{p^{-}(x_{\alpha})}p^{-}(x_{\alpha}\wedge y_{\beta})=p^{+}(x_{\alpha}\wedge y_{\beta}) \tag{1-83}
$$

又已知$\sum_{\alpha=1}^{2^m}p^{+}(x_{\alpha})=1$，代入前面式子得

$$
\frac{\partial G}{\partial w_{ij}}=-\frac{1}{T}[\sum_{\alpha=1}^{2^m}\sum_{\beta=1}^{2^r}\frac{p^{+}(x_{\alpha})}{p^{-}(x_{\alpha})}p^{+}(x_{\alpha}\wedge y_{\beta})x_i^{\alpha\beta}x_j^{\alpha\beta}-\sum_{\lambda=1}^{2^m}\sum_{\mu=1}^{2^r}p^{-}(x_{\lambda}\wedge y_{\mu})x_i^{\lambda\mu}x_j^{\lambda\mu}]=-\frac{1}{T}(p_{ij}^{+}-p_{ij}^{-})
$$

式中，p_{ij}^{+}表示网络受到学习样本的约束且系统达到平衡时第i和第j个神经网络元同时为1的概率；p_{ij}^{-}表示系统为自由状态且达到平衡时第i和第j个神经网络元同时为1的概率。

最后归纳出Boltzmann机网络的学习步骤如下。

（1）随机设定网络的连接权的初值$w_{ij}(0)$；

（2）按照已知概率$p^{+}(x_{\alpha})$，依次给定学习样本，在学习样本的约束下按照模拟退火程序运行网络直至到达平衡状态，统计出各p_{ij}^{+}。在无约束条件下按同样的步骤同样的次数运行网络，统计出各p_{ij}^{-}。

（3）按以下公式修改w_{ij}

$$
w_{ij}(k+1)=w_{ij}(k)+\alpha(p_{ij}^{+}-p_{ij}^{-}),\alpha>0
$$

重复以上步骤，直到$p_{ij}^{+}-p_{ij}^{-}$小于一定的容限。

1.2.11　神经网络的训练

根据前面介绍的几种典型神经网络可知，神经网络可应用于许多方面，它主要有以下一些特点。

（1）具有自适应功能

它主要是根据所提供的数据，通过学习和训练，找出和输出之间的内在联系，从而求得问题的解答，而不是依靠对问题的先验知识和规则，因而它具有很好的适应性。

（2）具有泛化功能

它能处理那些未经训练过的数据，而获得相对于这些数据的合适的解答。同样，它也能够处理那些有噪声或不完全的数据，从而显示了很好的容错能力。对于许多实际问题来说，泛化能力是非常有用的，原因是现实世界所获得的数据常常受到一定的污染或残缺不全。

（3）具有非线性映射功能

现实的问题常常是非常复杂的，各个因数之间互相影响，呈现出复杂的非线性关系，神经元网络为处理这些问题提供了有用的工具。

（4）高度并行处理

神经网络的处理是高度并行的，因此用硬件实现的神经网络的处理速度可远远高于通常计算机的处理速度。与常规的计算机程序相比较，神经网络主要基于所测量的数据对系统进行建模、估计和逼近，它可应用于如分类、预测及模式识别等众多方面。例如，函数映射是功能建模的一个典型例子。

传统的计算机程序也可完成类似的任务，在某些方面它们可以互相替代。然而更主要的是它们各有所长。传统的计算机程序比较适合于那些需要高精度的数值计算或者需要符号处理的那些任务。例如，财务管理和计算比较适合于采用计算机程序，而不适合于采用神经网络。对于那些几乎没有规则，数据不完全或者多约束优化问题，则适合于用神经网络。用神经网络来控制一个工业过程便是这样的例子，对于这种情况很难定义规则，历史数据很多而且充满噪声，准确的计算是毫无必要的。

在某些情况下，应用神经网络会存在严重的缺点。当所给数据不充分或不存在可学习的映射关系时，神经网络可能找不到满意的解。其次，有时很难估价神经网络给出的结果。神经网络中的连接权系数是千万次数据训练后的结果，对它的意义很难给出明确的解释，它对输出结果的影响也是非常复杂的。神经网络的训练是很慢的，而且有时需要付出很高的代价，这一方面是由于需要收集、分析和处理大量的训练数据，同时还需要相当的经验来选择合适的参数。

神经网络在实际应用时的执行时间也是需要加以检验的。执行时间取决于连接权的个数，它大体与网络节点数的平方成正比。因此网络节点的稍许增加便可能引起执行时间的增长。对于有些应用问题尤其是控制，太长的执行时间则可能阻碍它的实际应用。这种情况下必须采用专用的硬件。

总之，应根据实际问题的特点来确定是采用神经网络还是常规的计算机程序。这两者可以结合起来使用。例如，神经网络可用做一个大的应用程序中的一个组成部分，其作用类似于可调用的一个函数，应用程序将一组数据传给神经网络，神经网络将结果返回给应用程序。

下面讨论训练神经网络的具体步骤和几个实际问题。

1．产生数据样本集

为了成功地开发出神经网络，产生数据样本集是第一步，也是十分重要和关键的一步。这里包括原始数据的收集、数据分析、变量选择以及数据的预处理，只有经过这些步骤后，才能对神经网络进行有效的学习和训练。

首先要在大量的原始测量数据中确定出最主要的输入模式。例如，若两个输入具有很

强的相关性，则只需取其中一个作为输入，这就需要对原始数据进行统计分析，检验它们之间的相关性。又如，工业过程可能记录了大量的压力、温度和流量数据，这时就需要对它们进行相关分析，找出其中一两个最主要的量作为输入。

在确定了最重要的输入量后，需进行尺度变换和预处理。尺度变换常常将它们变换到[−1,1]或[0,1]的范围。在进行尺度变换前必须先检查是否存在异常点（或称野点），这些点必须剔除。通过对数据的预处理分析还可以检验其是否存在周期性、固定变化趋势或其他关系。对数据的预处理就是要使得经变换后的数据对于神经网络更容易学习和训练。例如，在过程控制中，采用温度的增量或导数比用温度值本身更能说明问题，也更容易找出变量之间的实质联系。在进行数据预处理时，主要应用信号处理或特征抽取技术，如计算数据的和、差、倒数、乘幂、求根、对数、平均、滑动平均以及傅里叶变换等。神经网络本身也可以作为数据预处理的工具，为另一个神经网络准备数据。

对于一个复杂问题，应该选择多少数据，这也是一个很关键的问题。系统的输入输出关系就包含在这些数据样本中。一般说来，取的数据越多，学习和训练的结果便越能正确反映输入输出关系。但是，选太多的数据将增加收集、分析数据以及网络训练所付出的代价。当然，选太少的数据则可能得不到正确的结果。事实上，数据的多少取决于许多因素，如网络的大小、网络测试的需要以及输入输出的分布等。其中，网络大小最关键。通常较大的网络需要较多的训练数据。一个经验规则是：训练模式应是连接权总数的5～10倍。

在神经网络训练完成后，需要有另外的测试数据来对网络加以检验，测试数据应是独立的数据集合。最简单的方法是：将收集到的可用数据随机地分成两部分，譬如说其中三分之二用于网络的训练，另外三分之一用于将来的测试，随机选取的目的是为了尽量减小这两部分数据的相关性。

影响数据大小的另一个因素是输入模式和输出结果的分布，对数据预先加以分类可以减少所需的数据量。相反，数据稀薄不匀甚至互相覆盖则势必要增加数据量。

2. 确定网络的类型和结构

在训练神经网络之前，首先要确定所选用的网络类型。神经网络的类型很多，需根据问题的性质和任务的要求来合适地选择网络类型。一般是从已有的网络类型中选用一种比较简单而又能满足要求的网络，新设计一个网络类型来满足问题的要求往往比较困难。

在网络的类型确定后，剩下的问题是选择网络的结构和参数。以BP网络为例，需选择网络的层数、每层的节点数、初始权值、阈值、学习算法、数值修改频度、节点变换函数及参数、学习率及动量项因子等参数。这里有些项的选择有一些指导原则，但更多的是靠经验和试凑。

对于具体问题，若确定了输入和输出变量后，网络输入层和输出层的节点个数也就随之确定了。对于隐含层的层数，可首先考虑只选择一个隐含层。剩下的问题是如何选择隐含层的节点数。其选择原则是：在能正确反映输入输出关系的基础上，尽量选取较少的隐含层

节点数，使网络尽量简单。具体选择可有如下两种方法：

（1）先设置较少的节点，对网络进行训练，并测试网络的逼近误差（后面还将介绍训练和测试的具体方法），然后逐渐增加节点数，直到测试的误差不再有明显的减小为止。

（2）先设置较多的节点，在对网络进行训练时，采用如下的误差代价函数

$$J_f=\frac{1}{2}\sum_{p=1}^{P}\sum_{i=1}^{N_Q}(t_{pi}-x_{pi})^2+\epsilon\sum_{q=1}^{Q}\sum_{i=1}^{N_Q}\sum_{j=1}^{N_{Q-1}}|w_{ij}^{(q)}|=J+\epsilon\sum_{q,i,j}|w_{ij}^{(q)}|$$

式中，J 仍与以前的定义相同，它表示输出误差的平方和。引入第二项的作用相当于引入一个“遗忘”项，其目的是为了使训练后的连接权系数尽量小。可以求得这时 J_f 对 w_{ij} 的梯度为

$$\frac{\partial J_f}{\partial w_{ij}^{(q)}}=\frac{\partial J}{\partial w_{ij}^{(q)}}+\varepsilon\,\mathrm{sgn}(w_{ij}^{(q)}) \tag{1-84}$$

利用该梯度可以求得相应的学习算法。利用该学习算法，在训练过程中只有那些确实必要的连接权才予以保留，而那些不很必要的连接权将逐渐衰减为零。最后可去掉那些影响不大的连接权和相应的节点，从而得到一个适当规模的网络结构。

若采用上述任一方法选择得到的隐含层节点数太多，则可考虑采用两个隐含层。为了达到相同的映射关系，采用两个隐含层的节点总数常常比只用一个隐含层时少。

3. 训练和测试

最后一步是对网络进行训练和测试，在训练过程中，训练样本数据需要反复地使用。对所有样本数据正向运行一次并反向传播修改连接权值一次称为一次训练（或一次学习），这样 的训练需要反复地进行下去直至获得合适的映射结果。通常，训练一个网络需要成百上千次。

特别应该注意的一点是，并非训练的次数越多，越能得到正确的输入输出的映射关系。训练网络的目的在于找出蕴含在样本数据中的输入和输出之间的本质联系，从而对于未经训练的输入也能给出合适的输出，即网络具备泛化功能。由于所收集的数据都是包含噪声的，训练的次数过多，网络将包含噪声的数据都记录了下来，在极端情况下，训练后的网络可以实现相当于查表的功能。但是，对于新的输入数据却不能给出合适的输出，即并不具备很好的泛化功能。网络的性能主要用它的泛化能力来衡量，它并不是用对训练数据的拟合程度来衡量，而是要用一组独立的数据来加以测试和检验。在用测试数据检验时，保持连接权系数不改变，只用该数据作为网络的输入，正向运行该网络，检验输出的均方误差。实际操作时训练和测试应该交替进行，即每训练一次，同时用测试数据测试一遍，画出均方误差随训练次数的变化曲线，如图 1-43 所示。

从误差曲线可以看出，在用测试数据检验时，均方误差开始逐渐减小，当训练次数再增加时，测试检验误差反而增加。误差曲线上极小点所对应的即为恰当的训练次数，若再训练，则为“过度训练”了。

对于网络隐含层节点数的选择，如果采用试验法，那么也必须将训练与测试相结合，最终也用测试误差来衡量网络的性能。均方误差与隐含层节点数也有与如图 1-43 所示相类似的关系，因此也不是节点数越多越好。

网络的节点数对网络的泛化能力有很大影响，节点数太多，它倾向于记住所有的训练数据，包括噪声的影响，反而降低了泛化能力；而节点数太少，它不能拟合样本数据，因而也谈不上有较好的泛化能力。选择节点数的原则是：选择尽量少的节点数以实现尽量好的泛化能力。

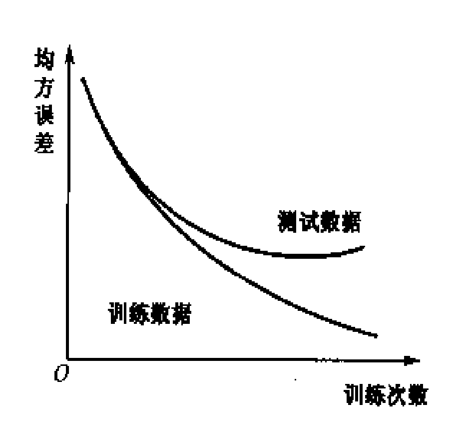


图 1-43 均方误差曲线

在用试验法选择其他参数时，也必须最终检验测试数据的误差。例如，初始权值的选择，一般可用随机法产生。为避免局部权值问题，可选取多组初始权值，最后选用最好的一种，也是靠检验测试数据误差来进行比较。

1.3 神经网络控制系统

神经网络发展至今已有半个多世纪的历史，概括起来经历了三个阶段：20 世纪 40—60 年代的发展初期；20 世纪 70 年代的研究低潮期；20 世纪 80 年代，神经网络的理论研究取得了突破性进展。神经网络控制是将神经网络在相应的控制系统结构中当做控制器或辨识器。神经网络控制的发展，虽仅有十余年的历史，但已有了多种控制结构。

1.3.1 神经控制的基本原理

传统的基于模型的控制方式，是根据被控对象的数学模型及对控制系统要求的性能指标来设计控制器，并对控制规律加以数学解析描述；模糊控制是基于专家经验和领域知识总结出若干条模糊控制规则，构成描述具有不确定性复杂对象的模糊关系，通过被控系统输出误差及误差变化和模糊关系的推理合成获得控制量，从而对系统进行控制。这两种控制方式都具有显式表达知识的特点，而神经网络不善于显式表达知识，但是它具有很强的逼近非线性函数的能力，即非线性映射能力。把神经网络用于控制正是利用它的这个独特优点。

众所周知，控制系统的目的在于通过确定适当的控制量输入，使得系统获得期望的输出特性。图 1-44（a）为一般反馈控制系统的原理图，图 1-44（b）采用神经网络替代图 1-44（a）中的控制器。为了完成同样的控制任务，下面来分析一下神经网络是如何工作的。

设被控制对象的输入 u 和系统输出 y 之间满足如下非线性函数关系，即

$$y = g(u) \tag{1-85}$$

控制的目的是确定最佳的控制量输入 u，使系统的实际输出 y 等于期望的输出 y_d。在该系统中，可把神经网络的功能看做输入输出的某种映射，或称函数变换，并设它的函数关系为

$$u = f(y_d) \tag{1-86}$$

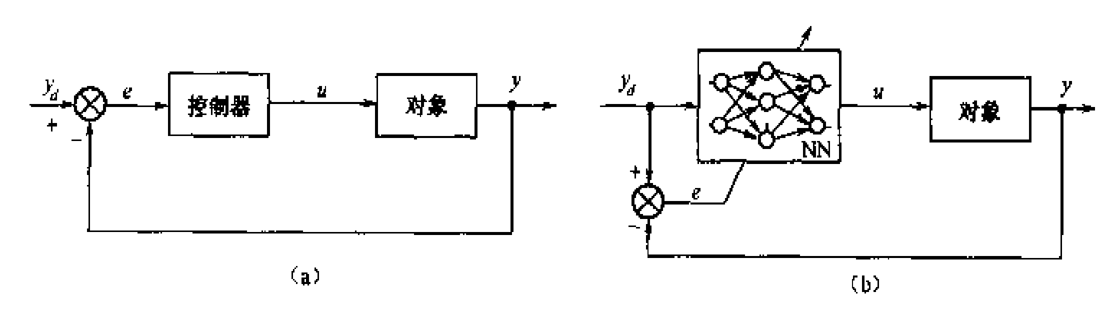


图1-44 反馈控制与神经网络

为了满足系统输出 y 等于期望的输出 y_d，将式（1-86）代入式（1-85），可得

$$y = g[f(y_d)] \tag{1-87}$$

显然，当 $f(\cdot)=g^{-1}(\cdot)$ 时，满足 $y=y_d$ 的要求。

由于要采用神经网络控制的被控对象一般是复杂的且多具有不确定性，因此非线性函数 $g(\cdot)$ 是难以建立的，可以利用神经网络具有逼近非线性函数的能力来模拟 $g^{-1}(\cdot)$，尽管 $g(\cdot)$ 的形式未知，但通过系统的实际输出 y 与期望输出 y_d 之间的误差来调整神经网络中的连接权值，即让神经网络学习，直至误差

$$e = y_d - y = 0 \tag{1-88}$$

的过程，就是神经网络模拟 $g^{-1}(\cdot)$ 的过程，它实际上是对被控对象的一种求逆过程。由神经网络的学习算法实现这一求逆过程，就是神经网络实现直接控制的基本思想。

1.3.2 神经网络在控制中的主要作用

由于神经网络是从微观结构与功能上对人脑神经系统的模拟而建立起来的一类模型，具有模拟人的部分智能的特性，主要是具有非线性、学习能力和自适应性，使神经控制能对变化的环境（包括外加扰动、量测噪声、被控对象的时变特性三个方面）具有自适应性，且成为基本上不依赖于模型的一类控制，因此决定了它在控制系统中应用的多样性和灵活性。

为了研究神经网络控制的多种形式，先来给出神经网络控制的定义。所谓神经网络控制，即基于神经网络的控制或简称神经控制，是指在控制系统中采用神经网络这一工具对难以精确描述的复杂的非线性对象进行建模，或充当控制器，或优化计算，或进行推理，或故障诊断，以及同时兼有上述某些功能的适应组合，将这样的系统统称为基于神经网络的控制系统，称这种控制方式为神经网络控制。

根据上述定义，可以将神经网络在控制中的作用分为以下几种：

（1）在基于精确模型的各种控制结构中充当对象的模型；

（2）在反馈控制系统中直接起控制器的作用；

（3）在传统控制系统中起优化计算作用；

（4）在与其他智能控制方法和优化算法，如模糊控制、专家控制及遗传算法等相融合中，为其提供非参数化对象模型、优化参数、推理模型及故障诊断等。

人工智能中的新技术不断出现及其在智能控制中的应用，必将使神经网络在和其他新技术的相融合中，在智能控制中发挥更大的作用。

神经网络控制主要是为了解决复杂的非线性、不确定、不确知系统在不确定、不确知环境中的控制问题，使控制系统稳定性好、鲁棒性强，具有满意的动静态特性。为了达到要求的性能指标，处在不确定、不确知环境中的复杂的非线性不确定、不确知系统的设计问题，就成了控制研究领域的核心问题。为了解决这类问题，可以在系统中设置两个神经网络，如图 1-45 所示。图中的神经网络 NNI 作为辨识器，由于神经网络的学习能力，辨识器的参数可随着对象、环境的变化而自适应地改变，故它可在线辨识非线性不确定、不确知对象的模型。辨识的目的是根据系统所提供的测量信息，在某种准则意义下估计出对象模型的结构和参数。图中的神经网络 NNC 作为控制器，其性能随着对象、环境的变化而自适应地改变（根据辨识器）。

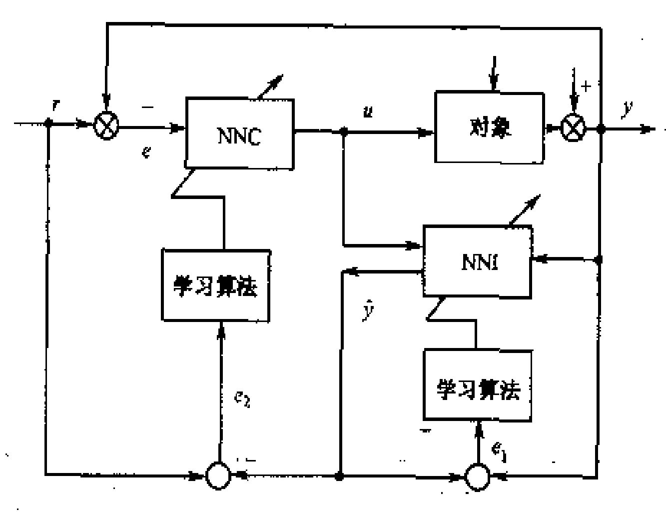


图 1-45　神经网络控制系统

在图 1-45 所示的系统中，对于神经控制系统的设计，就是对神经辨识器 NNI 和神经控制器 NNC 结构（包括神经网络种类、结构）的选择，并在一定的准则函数下，它们的权系数经由学习与训练，使之对应于不确定、不确知系统与环境，最后使控制系统达到要求的性能。由于该神经网络控制结构有两个神经网络，它是在高维空间搜索寻优，网络训练时，可调参数多，需调整的权值多，且收敛速度与所选的学习算法、初始权值有关，因此系统设计有相当难度。除了设计者所掌握的知识和经验外，还必须应用计算机硬件、软件技术作为神经网络控制设计的工具。

1.3.3　神经网络控制系统的分类

神经网络控制的结构和种类划分，根据不同观点可以有不同的形式，目前尚无统一的

分类标准。

1991 年，Werbos 将神经网络控制划分为学习控制、直接逆动态控制、神经自适应控制、BTT 控制和自适应决策控制五类。

1992 年，Hunt 等人发表长篇综述文章，将神经网络控制结构分为监督控制、直接逆控制、模型参考控制、内模控制、预测控制、系统辨识、最优决策控制、自适应线性控制、增强学习控制、增益排队论及滤波和预报等。

上述两种分类并无本质差别，只是后者划分更细一些，几乎涉及传统控制、系统辨识、滤波和预报等所有方面，这也间接地反映了随着神经网络理论和应用研究的深入，将向控制领域、信息领域等进一步渗透。

为了更能从本质上认识神经网络在实现智能控制中的作用和地位，1998 年，李士勇将神经网络控制从它与传统控制和智能控制两大门类的结合上考虑分为两大类：基于传统控制理论的神经控制和基于神经网络的智能控制。

1．基于传统控制理论的神经控制

将神经网络作为传统控制系统中的一个或几个部分，用以充当辨识器，或对象模型，或控制器，或估计器，或优化计算等。这种方式很多，常见的一些方式归纳如下：

1）神经直接逆动态控制

神经直接逆动态控制采用受控对象的一个逆模型，它与受控对象串联，以便使系统在期望响应（网络输入）与受控对象输出间得到一个相同的映射。因此，该网络直接作为前馈控制器，而且受控对象的输出等于期望输出。图 1-46 所示为神经直接逆动态控制的两种结构方案。

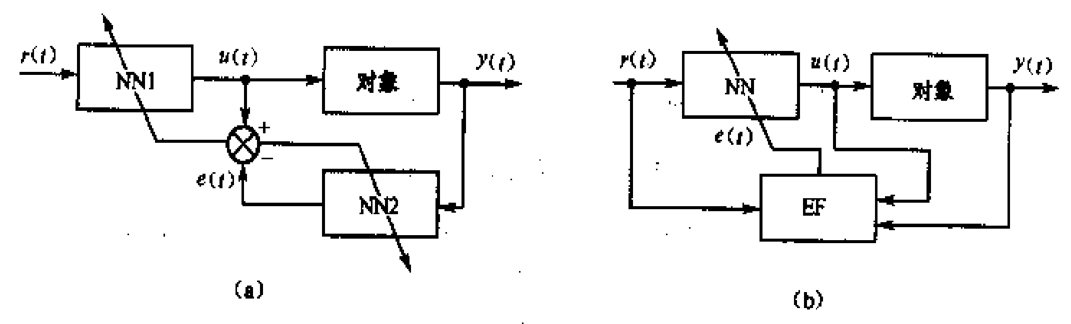


图 1-46 神经直接逆动态控制的两种结构方案

在图 1-46（a）中，有两个结构相同的神经网络 NN1 和 NN2，NN1 是神经网络前馈控制器，其输入是期望输出信号 $r(t)$，而输出控制信号 $u(t)$作用于对象；NN2 是神经网络逆辨识器，接收对象输出 $y(t)$，产生相应的输出。利用 NN1 和 NN2 两个神经网络输出的差值 $e(t)$，来同时调整 NN1 和 NN2 的连接权值，使这个差值 $e(t)$最终趋于零，做到 $y(t)\to r(t)$。神经网络控制器 NN1 和神经网络逆辨识器 NN2 具有相同的逆模型网络结构，而且采用同样的学习算法。

神经网络控制器 NN1 与被控对象 P 串联，实现对象 P 的逆模型 $\hat{P}^{-1}$，且能在线调整。

可见，这种控制结构要求对象动态可逆。若 $\hat{P}^{-1}=P^{-1}$，则 $\hat{P}^{-1}P=1$，在理论上可做到 $y(t)=r(t)$。输出 y 跟踪输入 r 的精度，取决于逆模型的精确程度。

下面分两种情况来分析 NN1 和 NN2 的运行情况。

（1）仅考虑 NN1 和对象而不考虑 NN2 的情况

如果目标是驱动输出 $y(t)$逼近 $r(t)$，则最好的策略是令 NN1 为对象的逆动态近似。对于未知对象的动态，没有一个简便的方法确定网络的正确权值，它将由对象逆动态的近似而求出。

（2）只考虑 NN2 和对象的情况

如果 NN2 的动态近似对象的逆动态，则差值 $e(t)$将为零。因此，一个好的控制策略就是用 NN2 去实现对象的逆近似。

对于未知对象，NN1 和 NN2 的参数将同时调整，当满足 $y(t)\to r(t)$时，NN1 和 NN2 将是对象逆动态的一个好的近似。

尽管作为控制器的逆模型参数可通过在线学习调整，以期把受控系统的鲁棒性提高到一定程度，但由于神经直接逆动态控制结构是开环控制，不能有效地抑制扰动，因此很少单独作用。图 1-46（b）为神经直接逆控制的另一种结构方案，NN 为被控对象的逆模型，EF 为评价函数。

2）神经自适应控制

神经自适应控制只是采用神经网络辨识对象模型，其余和传统形式自适应控制结构相同。

3）神经自校正控制

自校正控制属于自适应控制，将神经网络同自校正控制相结合，就构成了神经网络自校正控制。基于神经网络的自校正控制有两种结构：直接型与间接型。

（1）神经直接自校正控制

该控制系统由一个常规控制器和一个具有离线辨识能力的神经网络辨识器组成，由于神经网络的非线性函数的映射能力，使得它可以在自校正控制系统中充当未知系统函数逼近器，且具有很高的建模精度。神经直接自校正控制的结构基本上与直接逆动态控制相同。

（2）神经间接自校正控制

间接自校正控制一般称为自校正控制。自校正控制是一种利用辨识器将对象参数进行在线估计，用控制器实现参数的自动整定相结合的自适应控制技术，它可用于结构已知而参数未知但恒定的随机系统，也可用于结构已知而参数缓慢变化的随机系统。神经自校正控制结构如图 1-47 所示，它由一个自校正控制器和一个能够在线辨识的神经网络辨识器组成。自校正控制器与被控对象构成反馈回路，根据神经网络辨识器和控制器设计规则，以得到控制器的参数。可见，辨识器和自校正控制器的在线设计是自校正控制实现的关键。

一般地，为使问题简化，假设被控对象为如下所示的一阶单变量非线性系统，即

$$y(k+1)=g[y(k)]+\varphi[y(k)]u(k) \tag{1-89}$$

式中，$u(k)$和 $y(k)$分别为对象的输入和输出； $g[y(k)],\varphi[y(k)]$ 为非零函数。

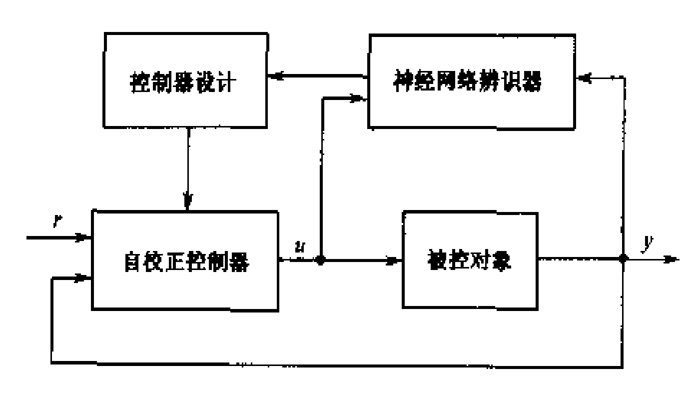


图 1-47 神经自校正控制系统

若 $g[y(k)],\varphi[y(k)]$ 已知，则根据确定性等价原则，控制器的控制律为

$$u(k)=\frac{r(k+1)-g[y(k)]}{\varphi[y(k)]} \tag{1-90}$$

此时，控制系统的输出 $y(k)$能精确地跟踪输入 $r(k)$。$r(k)$为系统的期望输出。

若 $g[y(k)],\varphi[y(k)]$ 未知，则可通过在线训练神经网络辨识器，使其逐渐逼近被控对象，由辨识器的 $Ng[y(k)],N\varphi[y(k)]$ 代替 $g[y(k)],\varphi[y(k)]$，则控制器的输出为

$$u(k)=\frac{r(k+1)-Ng[y(k)]}{N\varphi[y(k)]} \tag{1-91}$$

式中， $Ng[y(k)],N\varphi[y(k)]$ 为 $g[y(k)],\varphi[y(k)]$ 的估计值，为组成辨识器的非线性动态神经网络。

以上所描述的神经网络自校正控制系统，可进一步表示成如图 1-48 所示。

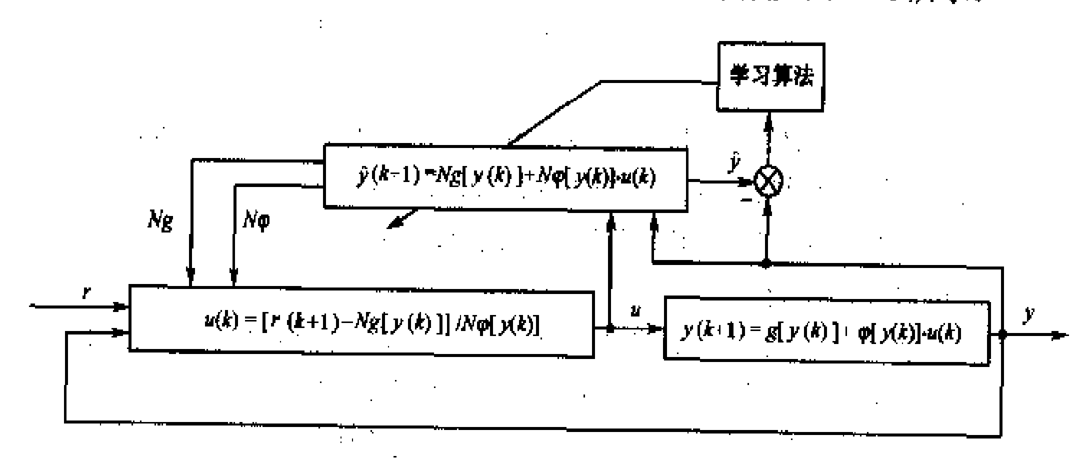


图 1-48 神经自校正控制框图

图 1-48 中神经网络辨识器

$$\hat{y}(k+1)=Ng[y(k)]+N\varphi[y(k)]u(k) \tag{1-92}$$

可由两个两层的 BP 网络实现，如图 1-49 所示。

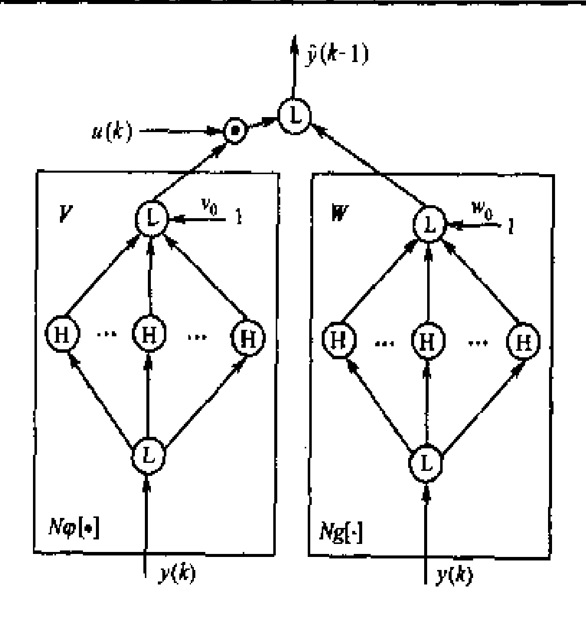


图 1-49　神经网络辨识器

图 1-49 中网络的输入为$\{y(k), u(k)\}$，输出为

$$\hat{y}(k+1)=Ng[y(k);W(k)]+N\varphi[y(k);V(k)]u(k) \tag{1-93}$$

式中，$W(k)=[w_0,w_1(k),w_2(k),\cdots,w_q(k)]$，$V(k)=[v_0,v_1(k),v_2(k),\cdots,v_q(k)]$ 分别为两个网络的权系数；q 是隐含层的非线性节点数，且有 $w_0=Ng[0,W]$，$v_0=N\varphi[0,V]$。

图 1-49 中的 L 为线性节点；H 为非线性节点，每一网络各 q 个，所用非线性作用函数为

$$f(x)=\frac{e^x-e^{-x}}{e^x+e^{-x}} \tag{1-94}$$

将式（1-91）代入式（1-89），则控制系统的输出为

$$y(k+1)=g[y(k)]+\varphi[y(k)]\frac{r(k+1)-Ng[y(k);W(k)]}{N\varphi[y(k);V(k)]} \tag{1-95}$$

可见，只有当$Ng[y(k);W(k)]\to g[y(k)],N\varphi[y(k);V(k)]=\varphi[y(k)]$时，才能使$y(k+1)\to r(k+1)$。

设神经网络学习的准则函数为

$$E(k)=\frac{1}{2}[r(k+1)-y(k+1)]^2=\frac{1}{2}e^2(k+1) \tag{1-96}$$

神经网络辨识器的训练过程，即权值的调整过程为

$$W(k+1)=W(k)+\Delta W(k),\quad V(k+1)=V(k)+\Delta V(k) \tag{1-97}$$

其中

$$\begin{cases}\Delta w_i(k)=-\eta_w\dfrac{\partial E(k)}{\partial w_i(k)}=-\eta_w\dfrac{\partial E(k)}{\partial y(k)}\dfrac{\partial y(k)}{\partial w_i(k)}\\[2ex]\Delta v_i(k)=-\eta_v\dfrac{\partial E(k)}{\partial v_i(k)}=-\eta_v\dfrac{\partial E(k)}{\partial y(k)}\dfrac{\partial y(k)}{\partial v_i(k)}\end{cases} \tag{1-98}$$

将式（1-96）和式（1-95）代入式（1-98）得

$$\Delta w_i(k) = -\eta_w \frac{\varphi[y(k)]}{N\varphi[y(k);V(k)]}\left\{\frac{\partial Ng[y(k);W(k)]}{\partial w_i(k)}\right\}e(k+1) \tag{1-99}$$

$$\Delta v_i(k) = -\eta_v \frac{\varphi[y(k)]}{N\varphi[y(k);V(k)]}\left\{\frac{\partial N\varphi[y(k);V(k)]}{\partial v_i(k)}\right\}e(k+1)u(k) \tag{1-100}$$

式（1-99）和式（1-100）中的$\partial Ng[y(k);W(k)]/\partial w_i(k)$和$\partial N\varphi[y(k);V(k)]/\partial v_i(k)$可仿照推导BP 算法过程计算。对于$\varphi[y(k)]$，由于对象特性未知，不能直接运算，但其符号已知，记为$\text{sgn}\{\varphi[y(k)]\}$，将其近似代替式（1-99）和式（1-100）中的$\varphi[y(k)]$，其正负可以确定该项在计算过程中收敛方向所起的作用。近似代替对权值变化所造成的误差，可通过调节η_w和η_v的大小进行补偿。此时，式（1-99）和式（1-100）可改写为

$$w_i(k+1) = w_i(k) - \eta_w \frac{\text{sgn}\{\varphi[y(k)]\}}{N\varphi[y(k);V(k)]}\left\{\frac{\partial Ng[y(k);W(k)]}{\partial w_i(k)}\right\}e(k+1) \tag{1-101}$$

$$v_i(k+1) = v_i(k) - \eta_v \frac{\text{sgn}\{\varphi[y(k)]\}}{N\varphi[y(k);V(k)]}\left\{\frac{\partial N\varphi[y(k);V(k)]}{\partial v_i(k)}\right\}e(k+1)u(k) \tag{1-102}$$

式中，$\eta_w > 0, \eta_v > 0$，它们决定神经网络辨识器收敛于被控对象的速度。

上述修正权值学习算法收敛时，所获得的控制律即为最佳的控制规律。

4）*神经模型参考自适应控制*

自校正控制和模型参考自适应控制是自适应控制中的两种重要形式，它们之间的差别在于自校正控制根据受控对象的正和（或）逆模型辨识结果直接调整控制器的内部参数，以期能够满足系统的给定性能指标。在模型参考自适应控制中，闭环控制系统的期望性能是由一个稳定的参考模型描述的，而该模型又是由输入/输出对$\{r(t), y_M(t)\}$确定的。它的控制目标在于使受控对象的输出$y(t)$与参考模型的输出$y_M(t)$渐近地匹配，即

$$\lim_{t\to\infty}\|y_M(t) - y(t)\| \leqslant \varepsilon, \varepsilon > 0$$

基于神经网络的模型参考自适应控制也有两种结构，即直接型与间接型。

（1）神经直接模型参考自适应控制

神经直接模型参考自适应控制系统如图 1-50 所示。该系统力图维持受控对象输出与参考模型输出间的差$e(t) = y(t) - y_M(t) \to 0$。但由于神经网络控制器反向传播需要已知受控对象的数学模型，当系统模型未知或部分未知时，神经网络控制器的学习与修正就很难进行。故对于不确定、不确知的对象，需采用神经间接参考自适应控制系统。

（2）神经间接模型参考自适应控制

神经间接模型参考自适应控制系统如图 1-51 所示，间接型比直接型多了一个神经网络辨识器 NNI，其余部分完全相同。神经网络辨识器 NNI 首先离线辨识受控对象的前馈模型，然后根据$e_1(t)$进行在线学习与修整。显然，NNI 能提供误差$e_2(t)$或者其变化率的反向传播。

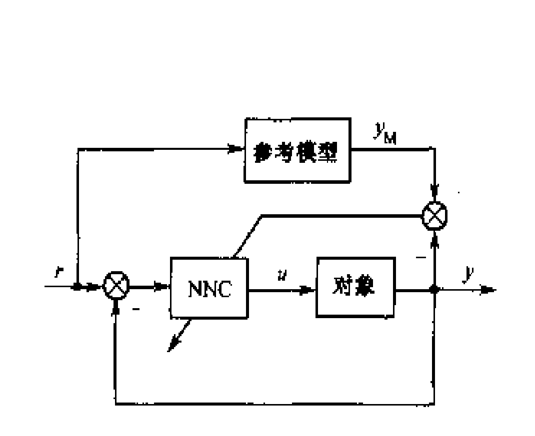


图 1-50　神经直接模型参考自适应控制系统

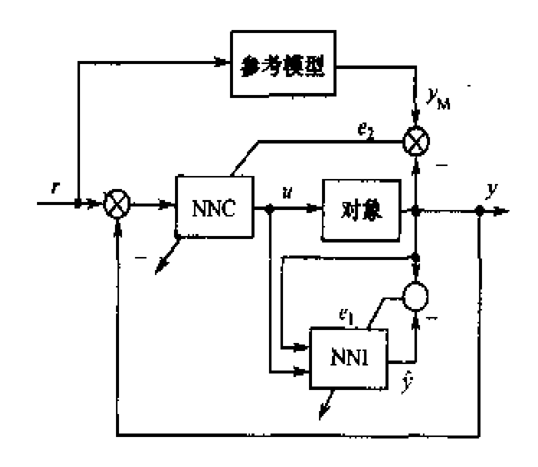


图 1-51　神经间接模型参考自适应控制系统

5）神经自适应 PID 控制

PID 控制是线性控制中的常用形式，原因是 PID 控制器结构简单、实现简易，且能对相当一些工业对象（或过程）进行有效的控制。但常规 PID 控制的局限性在于被控对象具有复杂的非线性特性时难以建立精确的数学模型，且由于对象和环境的不确定性，使控制参数整定困难，尤其是不能自调整，往往难以达到满意的控制效果。神经自适应 PID 控制是针对上述问题而提出的一种控制策略。采用神经网络调整 PID 控制参数就构成了神经网络自适应 PID 控制的结构，如图 1-52 所示。图中的 NN 为系统在线辨识器，系统在由 NN 对被控对象进行在线辨识的基础上，通过时时调整 PID 控制器的参数，使系统具有自适应性，达到有效控制的目的。

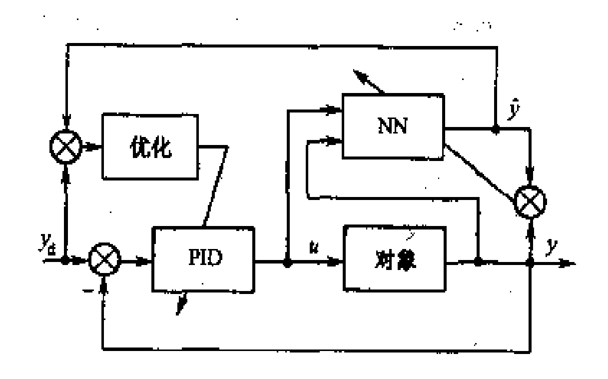


图 1-52　神经网络自适应 PID 控制系统

6）神经内模控制

内模控制（IMC，Internal Model Control）是由 Carcia 和 Morari 在 1982 年提出的，它具有结构简单、性能良好的优点。1986 年，Economou 等人将其推广到非线性系统，为非线性系统控制提供了有效的方法。IMC 经全面检验表明，其可用于鲁棒性和稳定性分析，而且是一种新的和重要的非线性系统控制方法，这种控制属于模型预测控制（MPC）的一种形式。无论是线性系统还是非线性系统，内模控制的原理是相同的。神经网络、模糊控制等智能控制理论和方法的引入，为非线性内模控制的研究开辟了新的途径。在传统的内模控制

结构中，用一个神经网络作为模型状态估计器，另一个神经网络作为控制器（或仍然采用常规控制器），就构成了神经内模控制的结构形式，如图 1-53 所示。

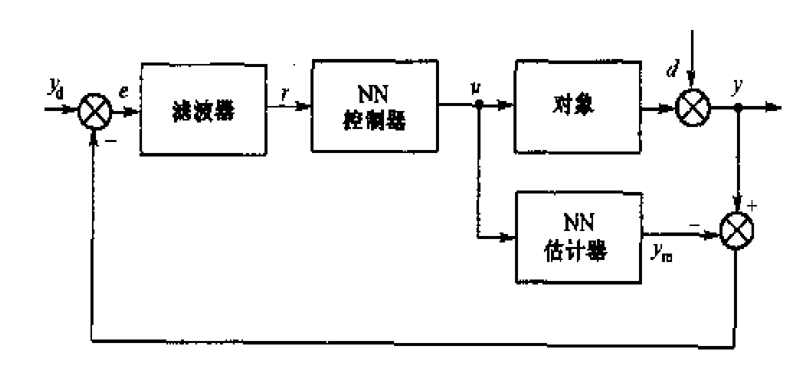


图 1-53 神经网络内模控制系统的结构

在图 1-53 中，神经网络估计器（NN 估计器）作为被控对象的近似模型与实际对象并行设置，系统输出与神经网络估计器输出间的差值用于反馈作用，同期望的给定值之差经一线性滤波器处理后，送给 NN 控制器（在正向控制通道上一个具有逆模型的神经网络控制器），然后由 NN 控制器经过多次训练，将间接地学习到对象的逆动态特性，此时，系统误差将趋于零。神经网络控制器与对象的逆有关。神经网络估计器也是基于神经网络的，但具有对象的正向模型。NN 估计器用于充分逼近被控对象的动态模型，NN 控制器不是直接学习被控对象的逆动态模型，而是以充当状态估计器的神经网络模型（内部模型）作为训练对象，间接地学习被控对象的逆动态特性。这样就回避了要估计 $\partial y(k+1)/\partial u(k)$ 而造成的困难。图中的滤波器通常为一线性滤波器，而且可被设计成满足必要的鲁棒性和闭环系统跟踪响应。

7）神经预测控制

预测控制是一种基于模型的控制，它是 20 世纪 70 年代发展起来的一种新的控制算法，具有预测模型、滚动优化和反馈校正等特点。已经证明该控制方法对于非线性系统能够产生希望的稳定性。图 1-54 为神经网络预测控制系统的结构，图中的 NNM 为神经网络对象响应预报器，NNC 为神经网络控制器。NNM 提供的预测数据送入优化程序，使性能目标函数在选择合适的控制信号 u 条件下达最小值，即

$$J=\sum_{j=1}^{n}[y_m(k+j)-y_r(k+j)]^2+\sum_{j=1}^{m}\lambda[u(k+j-1)-u(k+j-2)]^2$$

式中，n 为预测时域长度；m 为控制时域长度；λ 是控制加权因子；$u(k)$为控制信号；y_r 为期望响应；y_m 为网络模型响应。

8）神经最优决策控制

在最优决策控制系统中，状态空间根据不同控制条件被分成特征空间区域，控制曲面的实现是通过训练过程完成的。由于时间最优曲面通常是非线性的，因此有必要使用一个能够逼近非线性的结构。一种可能的方法是将状态空间量化成基本的超立方体，在这个立方体

中，控制作用是一个假设的常数。这个过程可由一个 LVQ 结构实现，很有必要让另一个网络充当分类器，若需要连续信号，则可以使用标准的反向传播结构，如图 1-55 所示。

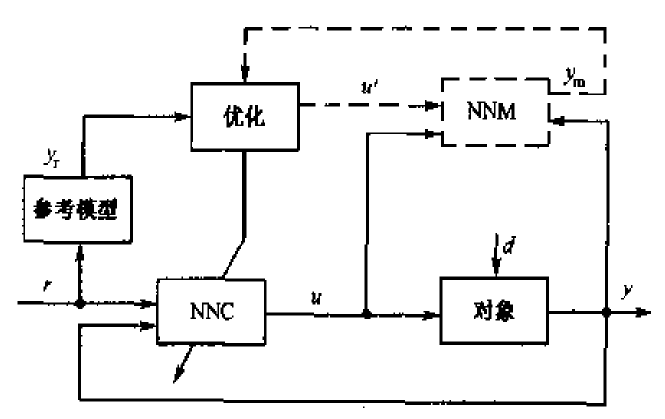


图 1-54　神经网络预测控制系统的结构

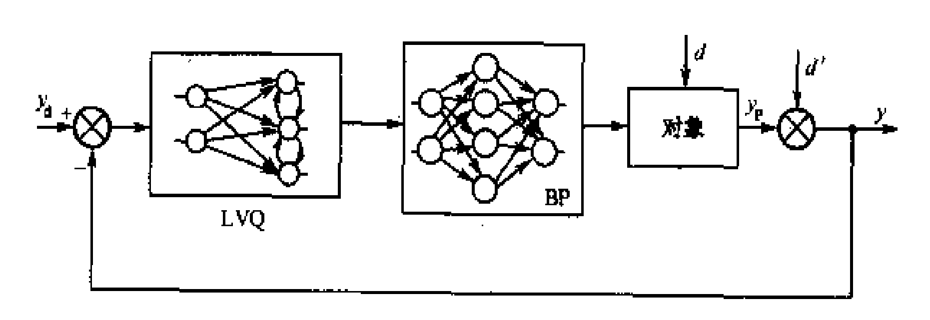


图 1-55　神经网络内模控制系统的结构

转换曲面不是已知的，而是经过在状态空间中训练点集来隐含定义的，这个状态空间的最优控制作用是已知的，在训练过程中，学习算法只根据目前指示给它的训练样本向量，在它的权中与所需要的控制条件一起进行调整。训练样本向量按顺序几次提供给控制器，直到在训练集中所有的样本向量都被正确地分类，或者分类误差已经达到某一稳态值。

神经网络与线性控制的结合还有其他的结构形式，如神经网络与常规反馈控制的结合，如图 1-56 所示。在神经网络的学习阶段采用常规控制，学习结束后，常规控制不再起作用，由神经网络控制器来控制。

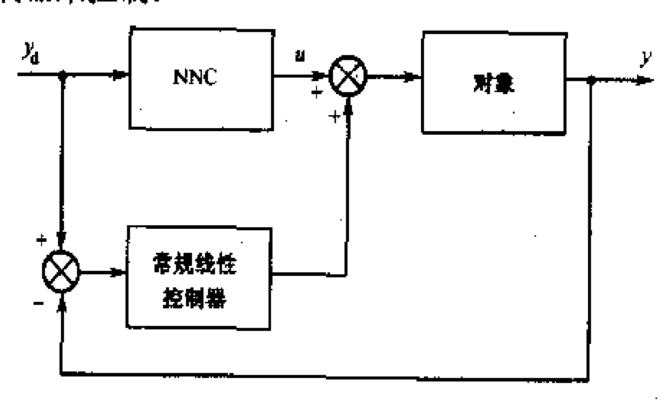


图 1-56　神经网络与常规反馈控制的结合控制系统

2. 基于神经网络的智能控制

基于神经网络的智能控制是只由神经网络单独进行控制或由神经网络同其他智能控制方式相融合的控制的统称，前者称神经控制，后者可称为神经智能控制。属于这一大类的有以下形式：

1）神经网络直接反馈控制

神经网络直接反馈控制是神经网络直接作为控制器，利用反馈和使用遗传算法进行自学习控制。这是一种只使用神经网络实现的一种智能控制方式。

2）神经网络专家系统控制

专家系统善于表达知识和逻辑推理，神经网络长于非线性映射和直觉推理，将二者相结合发挥各自的优势，就会获得更好的控制效果。

图 1-57 是一种神经网络专家系统的结构方案，这是一种将神经网络和知识基系统相结合用于智能机器人的控制系统结构。EC 是对动态系统 P 进行控制的基于规则的专家控制器，神经网络控制器 NC 将接收小脑模型关联控制器 CMAC 的训练，每当运行条件变化使神经控制器性能下降到某一限度时，运行监控器 EM 将调整系统工作状态，使神经网络处于学习状态，此时，EC 将保证系统的正常运行。该系统运行共有三种状态：EC 单独运行、EC 和 NC 同时运行、NC 单独运行，监控器 EM 负责管理它们之间运行的切换。

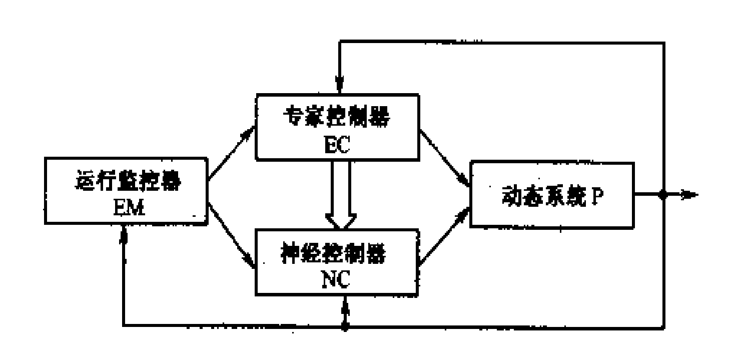


图 1-57　神经网络专家系统的结构

对复杂系统可采用递阶分级控制结构，如图 1-58（a）所示，下层为 NC，上层为 EC，利用 NC 的映射能力和运算能力进行实时控制，EC 则用于知识推理、决策、规划和协调。图 1-58（b）为一种分级结构，EC1 帮助 NC 进行训练等；NC 用于决策、求解问题；EC2 用来解释 NC 的输出结果并驱动执行机构对系统 P 进行控制。图 1-58（c）所示是利用神经网络控制完成专家系统中最耗费时间的模式匹配工作，以利于加速专家系统的执行。由上面结构不难看出，神经控制和专家系统的结合具有这样的特点：在分层结构中，EC 在上层，NC 在下层；在分级结构中，EC 在前级，NC 在后级。

3）神经网络模糊逻辑控制

模糊逻辑具有模拟人脑抽象思维的特点，而神经网络具有模拟人脑形象思维的特点，将二者相结合将有助于从抽象和形象思维两方面模拟人脑的思维特点，是目前实现智能控制

的重要形式。

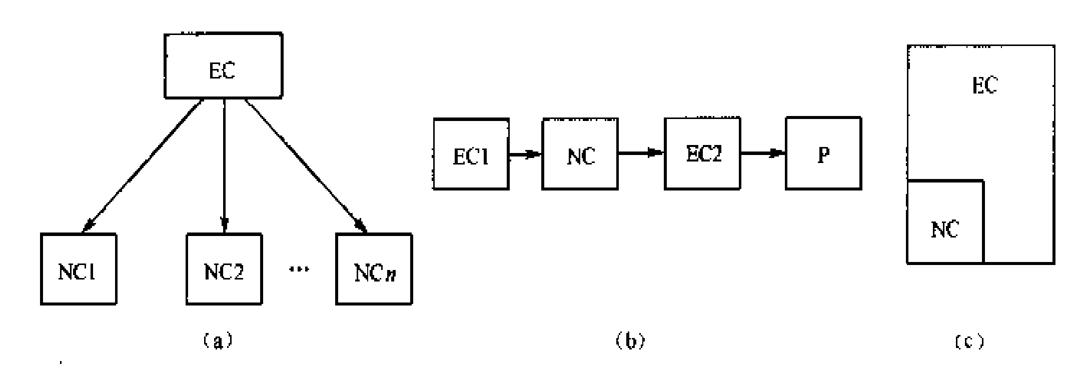


图 1-58　神经网络专家系统的递阶分级结构

模糊系统善于直接表示逻辑，适于直接表示知识；神经网络长于学习，通过数据隐含表达知识。前者适于自上而下的表达，后者适于自下而上的学习过程，二者存在一定的互补、关联性。因此，它们的融合可以取长补短，可以更好地提高控制系统的智能性。

神经网络和模糊逻辑相结合有以下几种方式：

（1）用神经网络驱动模糊推理的模糊控制

这种方法是利用神经网络直接设计多元的隶属函数，把 NN 作为隶属函数生成器组合在模糊控制系统中。

（2）用神经网络记忆模糊规则的控制

通过一组神经元不同程度的兴奋表达一个抽象的概念值，由此将抽象的经验规则转化成多层神经网络的输入/输出样本，通过神经网络，如 BP 网络记忆这些样本，控制器以联想记忆方式使用这些经验，在一定意义上与人的联想记忆思维方式接近。

（3）用神经网络优化模糊控制器的参数

在模糊控制系统中，对控制性能有影响的因素，除上述的隶属函数、模糊规则外，还有控制参数，如误差、误差变化的量化因子及输出的比例因子，都可以调整。利用神经网络的优化计算功能可优化这些参数，改善模糊控制系统的性能。

4）神经网络滑模控制

变结构控制从本质上应该看做是一种智能控制，将神经网络和滑模控制相结合就构成神经网络滑模控制。这种方法是将系统的控制或状态分类，根据系统和环境的变化进行切换和选择，利用神经网络具有的学习能力，在不确定的环境下通过自学习来改进滑模开关曲线，进而改善滑模控制的效果。

第 2 章　MATLAB 神经网络工具箱函数

利用神经网络能解决许多用传统方法无法解决的问题。神经网络在很多领域中都有应用，可实现各种复杂的功能。这些领域包括商业及经济估算、自动检测和监视、计算机视觉、语音处理、机器人和自动控制、优化问题、航空航天、银行金融业、工业生产等。神经网络是一门发展很快的学科，其应用领域也会随着其发展有更大的拓宽。本章将介绍 MATLAB 神经网络工具箱的应用。MATLAB 神经网络工具箱提供了丰富的演示实例，用 MATLAB 语言构造了典型神经网络的激活函数，编写了各种网络设计与训练的子程序。网络的设计者可以根据自己的需要去调用工具箱中有关神经网络的设计训练程序，使自己能够从繁琐的编程中解脱出来。

MATLAB 神经网络工具箱提供了许多进行神经网络设计和分析的工具函数，这给用户带来了极大的方便，即使不了解算法的本质，也可以直接应用功能丰富的函数来实现自己的目的。有关这些工具函数的使用可以通过 help 命令得到。本章将对这些函数的功能、调用格式，以及使用方法进行详细的介绍。随着 MATLAB 软件版本的提高，其对应的神经网络工具箱的内容也越来越丰富。它包括了很多现有的神经网络的新成果，涉及的网络模型有感知机神经网络、线性神经网络、BP 神经网络、径向基神经网络、自组织神经网络、学习向量量化神经网络、Elman 神经网络、Hopfield 神经网络等。

神经网络工具箱提供了很多经典的学习算法，使用它能够快速地实现对实际问题的建模求解。由于其编程简单，故使使用者节省了大量的编程时间。使用者可以把更多的精力投入到网络的设计而不是投入在具体程序的实现上。

2.1　感知机神经网络工具箱函数

MATLAB 神经网络工具箱提供了大量的与感知机相关的函数。在 MATLAB 工作空间的命令行中输入“help percept”，便可得到与感知机（Perceptron）相关的函数，进一步利用 help 命令又能得到相关函数的详细介绍。表 2-1 列出了感知机神经网络函数的名称和基本功能。

表 2-1　感知机神经网络的函数名称和基本功能

名　称	功　能
mae()	平均绝对误差性能函数
hardlim()	硬限幅传输函数
hardlims()	对称硬限幅传输函数

续表

名　称	功　能
plotpv( )	在坐标图上绘出样本点
plotpc( )	在已绘制的图上加分类线
initp( )	对感知机神经网络进行初始化
trainp( )	训练感知机神经网络的权值和偏值
trainpn( )	训练标准化感知机的权值和偏值
simup( )	对感知机神经网络进行仿真
learnp( )	感知机的学习函数
learnpn( )	标准化感知机的学习函数
newp( )	生成一个感知机
netsum( )	输入求和函数
train( )	神经网络训练函数
adapt( )	神经网络自适应训练函数
sim( )	神经网络仿真函数
init( )	初始化一个神经网络

1. 平均绝对误差性能函数 mae( )

感知机神经网络的学习规则为调整网络的权值和偏值，使网络的平均绝对误差和最小。平均绝对误差性能函数的调用格式为

perf=mae(E,w,pp)

式中，E 为误差矩阵或向量（E=T−Y），T 表示网络的目标向量，Y 表示网络的输出向量；w 为所有权值和偏值向量（可忽略）；pp 为性能参数（可忽略）；perf 表示平均绝对误差和。

2. 硬限幅传输函数 hardlim( )

硬限幅传输函数 hardlim( )可通过计算网络的输入得到该层的输出。如果网络的输入达到门限，则硬限幅传输函数的输出为 1；否则，为 0。这表明神经元可用来做出判断或分类。其调用格式为

a=hardlim(N)

或　　a=hardlim(Z,b)

a=hardlim(P)

函数 hardlim(N)在给定网络的输入矢量矩阵 N 时，返回该层的输出矢量矩阵 a。当 N 中的元素大于等于零时，返回的值为 1；否则，为 0。函数 hardlim(Z,b)用于矢量为成批处理且偏差存在的情况下，此时的偏差 b 和加权输入矩阵 Z 是分开传输的。偏差矢量 b 加到 Z 中的每个矢量中形成网络输入矩阵。返回的元素 a 是 1 还是 0，取决于网络输入矩阵中的元素是大于等于 0 还是小于 0。函数 hardlim(P)包含传输函数的特性名并返回问题中的特性。下面的特性可从任何传输函数中获得：

（1）delta——与传输函数相关的delta函数；

（2）init——传输函数的标准初始化函数；

（3）name——传输函数的全称；

（4）output——包含有传输函数最小、最大值的二元矢量。

例如，利用以下MATLAB命令可得如图2-1所示的硬限幅传输函数曲线。

```
>>N=-5:0.1:5;a=hardlim(N);plot(N,a)
```

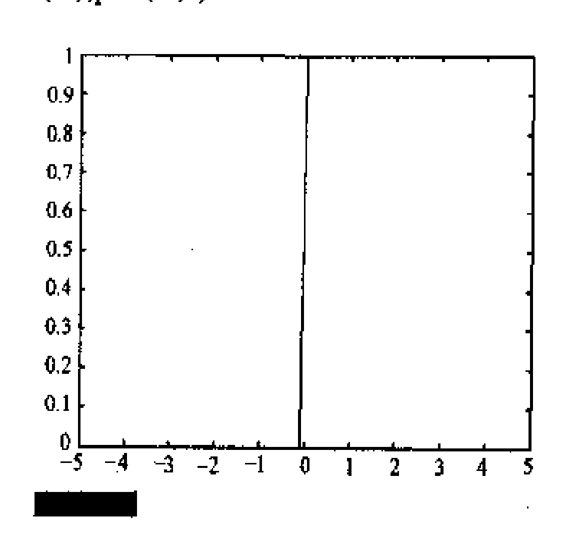


图2-1 硬限幅传输函数曲线

3．对称硬限幅传输函数hardlims( )

对称硬限幅传输函数 hardlims( )可通过计算网络的输入得到该层的输出。如果网络的输入达到门限，则硬限幅传输函数的输出为1；否则，为-1。例如，

```
>> w=eye(3);b=-0.5*ones(3,1);X=[1 0;0 1;1 1];a=hardlims(w*X,b)
```

其结果显示为

```
a =
     1    -1
    -1     1
     1     1
```

例2-1 建立一个感知机神经网络，使其能够完成“或”的功能。

解：为了完成“或”函数，建立一个两输入、单输出的一个单层感知机神经网络。根据表1-1中“或”函数的真值表，可得训练集的输入矩阵为X=[0 0 1 1;0 1 0 1]，目标向量为T=[0 1 1 1]。激活函数取硬限幅传输函数。

根据感知机学习算法的计算步骤，利用MATLAB的神经网络工具箱的有关函数编写的程序如下：

```
%感知机神经网络的第一阶段学习期（训练加权系数Wij）
```

```
err_goal=0.001;                    %给定期望误差最小值
max_epoch=5000;                    %给定训练最大次数
X=[0 0 1 1;0 1 0 1];T=[0 1 1 1];   %提供四组 2 输入 1 输出的训练集和目标值
%初始化 Wij(M-输入节点 j 的数量,L—输出节点 i 的数量, N—为训练集对数量)
[M,N]=size(X);[L,N]=size(T);
Wij=rand(L,M);b1=zeros(L,1);       %随机给定输出层的权值和偏值
for epoch=1:max_epoch
    y=hardlim(Wij*X,b1);           %计算网络输出层的各神经元输出
    E=T-y;                         %计算网络误差
    SSE=mae(E);                    %计算网络权值修正后的绝对误差
    if (SSE<err_goal) break;end
    Wij=Wij+E*X';                  %调整输出层加权系数
    b1=b1+E;                       %调整输出层的偏值
end
epoch, Wij                         %显示计算次数和加权系数
%感知机的第二阶段工作期(根据训练好的 Wki,Wij 和给定的输入计算输出)
X1=X;                              %给定输入
y=hardlim(Wij*X1,b1)               %计算网络输出层的各神经元输出
```

其结果显示为

```
epoch =
     3
Wij =
    1.5028    1.7095
y =
     0   1   1   1
```

4. 绘制样本点的函数 plotpv( )

利用 plotpv( ) 函数可在坐标图中绘出已知给出的样本点及其类别，不同的类别使用了不同的符号。其调用格式为

plotpv(X,T)

式中，X 定义了 n 个 2 或 3 维的样本，是一个 $2\times n$ 维或 $3\times n$ 维的矩阵；T 表示各样本点的类别，是一个 n 维的向量。如果 T 只含一元矢量，则目标为 0 的输入矢量画为“o”；目标为 1 的输入矢量画为“+”。如果 T 含二元矢量，则输入矢量对应如下：[0 0]用“o”；[0 1]用“+”；[1 0]用“*”；[1 1]用“×”。例如，MATLAB 命令如下：

```
>>X=[-0.5,-0.5,0.3,-0.1,0.2,0.0,0.6,0.8;-0.5,0.5,-0.5,1.0,0.5,-0.9,0.8,-0.6];
>>T=[1 1 0 1 1 0 1 0]; plotpv(X,T)
```

对样本不同的类别使用了不同的符号，如图 2-2 所示。

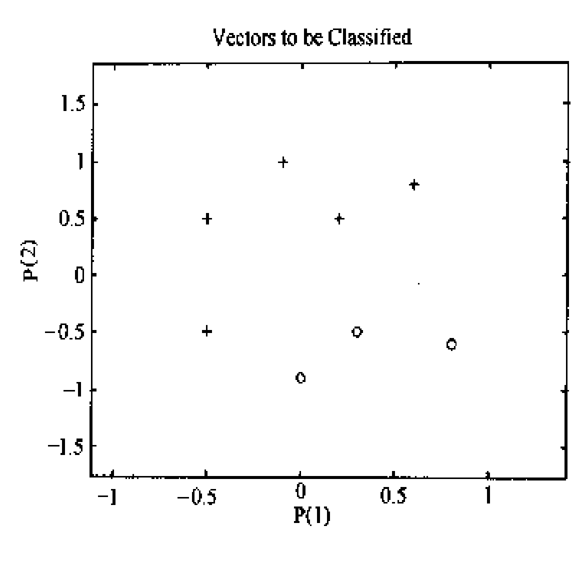


图 2-2 样本的分类

5. 在存在的图上画感知机的分类线函数 plotpc( )

硬特性神经元可将输入空间用一条直线（如果神经元有两个输入），或用一个平面（如果神经元有三个输入），或用一个超平面（如果神经元有三个以上输入）分成两个区域。plotpc(w,b)对含权矩阵 w 和偏差矢量 b 的硬特性神经元的两个或三个输入画一个分类线。这一函数返回分类线的句柄以便以后调用。plotpc(w,b,h)包含从前的一次调用中返回的句柄。它在画新分类线之前，删除旧线。

6. 感知机神经网络的初始化函数 initp( )

利用 initp( )函数可建立一个单层（一个输入层和一个输出层）感知机神经网络。其调用格式为

[W,b]=initp(R,S)

或

[W,b]=initp(X,T)

式中，R 为输入个数；S 为输出神经元数；W 为网络的初始权值；b 为网络的初始偏值。另外，R 和 S 可以用对应的输入向量矩阵 X 和目标向量 T 来代替，此时输入个数和输出神经元数根据 X 和 T 中的行数来设置。例如，利用以下命令可得到如图 2-3 所示的输入样本加网络初始分类线。

```
>>X=[0 0 1 1;0 1 0 1];T=[0 1 1 1];
>>[W,b]=initp(X,T);
>>plotpv(X,T);plotpc(W,b);
```

使用 plotpc( )函数可以在已绘制的图上加上感知机分类线（MATLAB6.1 及以下版本利用此命令不能产生分类线）。由图 2-3 可见，经过初始化后的网络对输入样本还不能正确进

行分类。

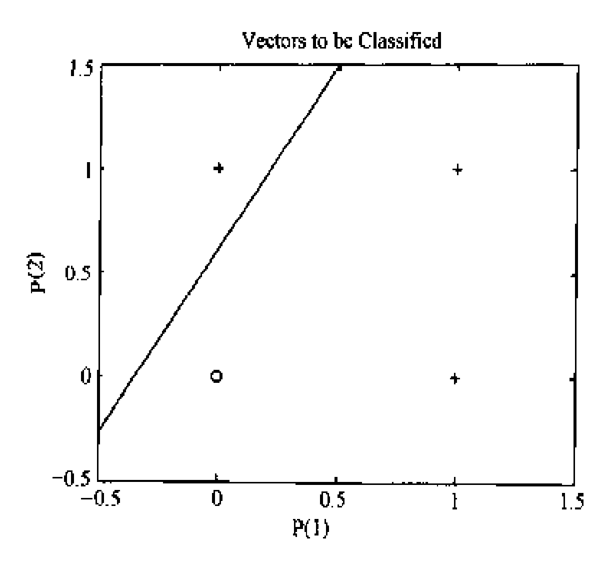


图 2-3　输入样本加网络初始分类线

7．用感知机准则训练感知机的函数 trainp( )

经过初始化建立的感知机还必须经过训练才能够实际应用。通过训练可决定网络的权值和偏值。对于感知机，其训练过程为：对于给定的输入向量，计算网络的实际输出，并与相应的目标向量进行比较，得到误差δ，然后根据相应的学习规则调整权值和偏值。重新计算网络在新的权值和偏值作用下的输出，重复上述的权值和偏值的调整过程，直到网络的输出与期望的目标向量相等或者训练次数达到预定的最大次数时才停止训练。之所以要设定最大的训练次数，是因为对于有些问题，使用感知机神经网络时是不能解决的。这正是感知机的缺点。训练感知机神经网络 trainp( )函数的调用格式为

[W,B,epochs,errors]= trainp(w,b,X,T,tp)

式中，w 为网络的初始权值；b 为网络的初始偏值；X 为网络的输入向量矩阵；T 表示网络的目标向量；tp=[disp_freq max_epoch]是训练控制参数，其作用是设定如何进行训练，其中 disp_freq 或 tp(1)是更新显示的迭代次数，默认值为 1；max_epoch 或 tp(2) 是训练的最大迭代次数，默认值为 100，如果给出了 tp，则任何参数的遗漏或 NaN 值都会使参数设定到默认值；W 为网络训练后的权值；B 为网络训练后的偏值；epochs 表示训练步数；errors 表示误差。例如，利用以下命令可得如图 2-4 所示的样本及分类线。

```
>>X=[0 0 1 1;0 1 0 1];T=[0 1 1 1]; [W,b]=initp(X,T); tp=[1 20];
>>[W,b,epochs,errors]=trainp(W,b,X,T,tp);
>>plotpv(X,T);plotpc(W,b);
```

由图 2-4 可知，经过训练后的网络，已能对输入样本进行正确分类。

用标准化感知机准则训练感知机的函数 trainpn( )的用法与函数 trainp( )相同，即使输入矢量的长度不同，使用标准化感知机准则也能使得学习过程收敛很快。

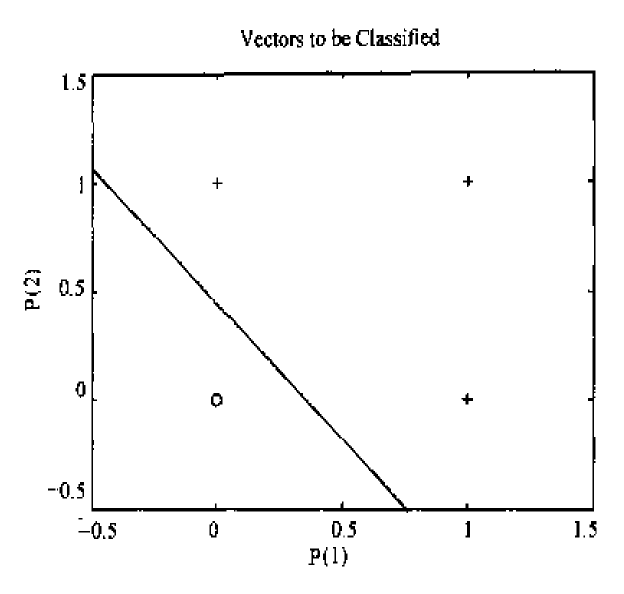


图 2-4 样本及分类线

8. 感知机神经网络的仿真函数 simup( )

神经网络一旦训练完成，网络的权值和偏值就已经确定了，于是就可以使用它来解决实际问题了。感知机由一系列硬特性神经元组成，运行速度很快，对简单的分类很有用。利用 simup( )函数可以测试一个感知机神经网络的性能。其调用格式为：

Y=simup(X,w,b)

式中，X 为网络的输入向量矩阵；w 为网络的权值；b 为网络的偏值；Y 表示网络的输出向量。

例 2-2 利用 trainp( ) 函数训练一个感知机神经网络，使其能够完成“或”的功能。

解：根据神经网络工具箱函数编写的程序如下：

```
X=[0 0 1 1;0 1 0 1];T=[0 1 1 1];          %提供四组2输入1输出的训练集和目标值
plotpv(X,T);                              %绘制输入样本的分类
[W,b]=initp(X,T);                         %初始化网络
figure;plotpv(X,T);plotpc(W,b);           %绘制输入样本加网络初始分类线
%训练网络,同时绘制输入样本加网络训练后的分类线
figure;
[W,b,epochs,errors]=trainp(W,b,X,T,-1);
figure;ploterr(errors);                   %绘制误差曲线
X1=X; y=simup(X1,W,b)                     %仿真网络，并给出输出值
```

执行以上程序后可得如下结果，如图 2-5～图 2-8 所示。

```
y =
    0  1  1  1
```

由以上结果及如图 2-7 所示可知，训练后的网络已具有“或”的功能，且可对输入样本

进行正确分类。

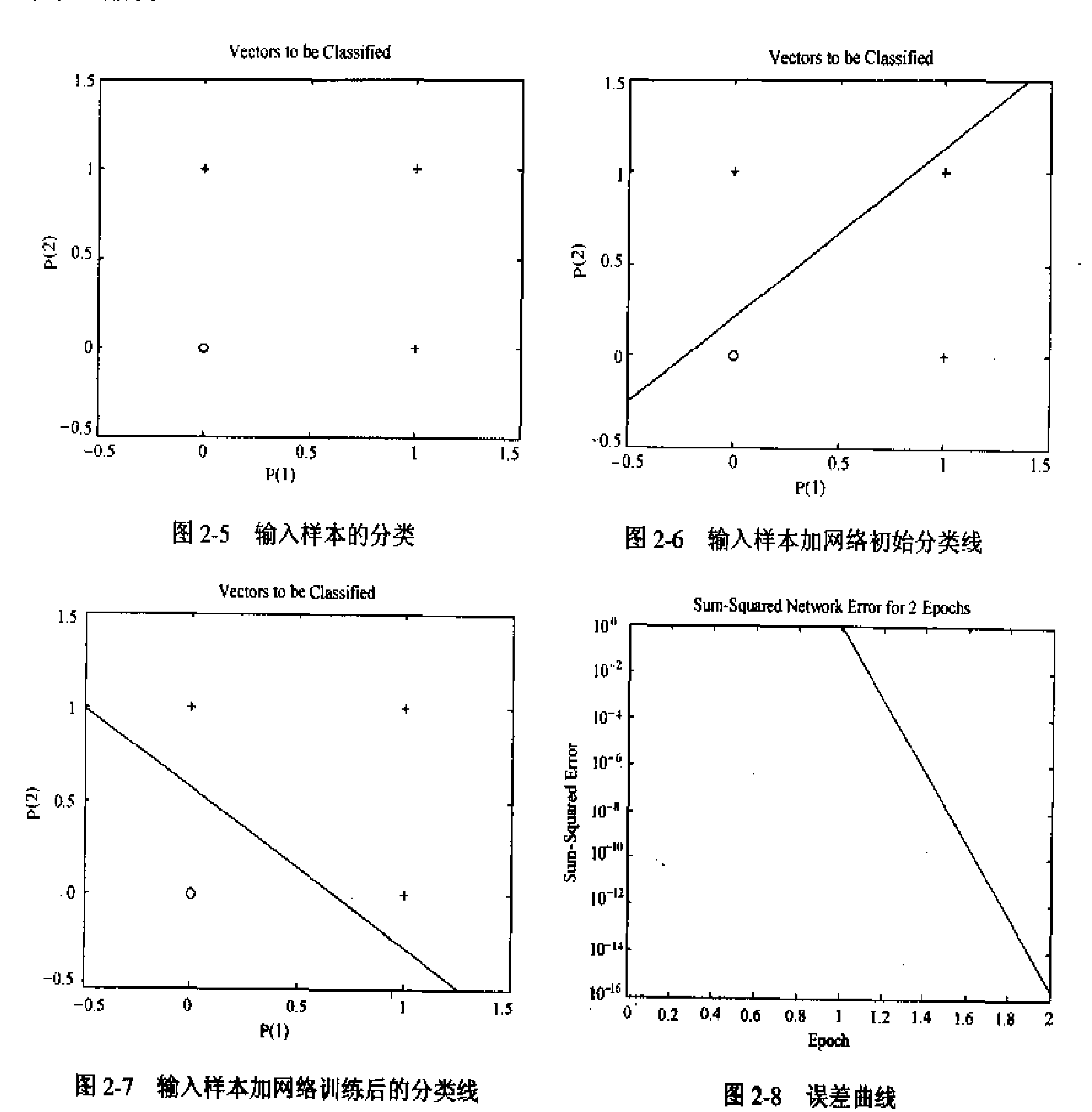


图 2-5 输入样本的分类

图 2-6 输入样本加网络初始分类线

图 2-7 输入样本加网络训练后的分类线

图 2-8 误差曲线

9. 感知机学习函数 learnp( )

感知机神经网络学习规则为调整网络的权值和偏值使网络平均绝对误差性能最小，以便对网络输入矢量正确分类。感知机的学习规则只能训练单层网络。该函数的调用格式为：

[dW, db]=learnp(X,E)

式中，X 为输入向量矩阵；E 为误差向量（E=T−Y）；T 表示网络的目标向量；Y 表示网络的输出向量；dW 为权值变化阵；db 为偏值变化阵。

例 2-3 利用 learnp( ) 函数学习一个感知机网络，使其同样能够完成“或”的功能。

解：根据神经网络工具箱函数编写的程序如下：

```
err_goal=0.001;max_epoch=10000;      %设置期望误差最小值和训练的最大次数
X=[0 0 1 1;0 1 0 1];T=[0 1 1 1];
[W,b]=initp(X,T);
for epoch=1:max_epoch
  y=simup(X,W,b);
  E=T-y;SSE=mae(E);                  %计算网络权值修正后的平均绝对误差
  if (SSE<err_goal) break;end
  [dW,db]=learnp(X,E);               %调整输出层加权系数和偏值
  W=W+dW;b=b+db;
end
epoch,W,y                            %显示计算次数，加权系数及网络输出
```

其结果显示为

```
epoch =
      5
W =
   1.3626   1.7590
y =
   0   1   1   1
```

10. 标准化感知机学习函数 learnpn( )

感知机学习规则在调整网络的权值和偏值时可利用下式，即

$$\Delta w = X(T-Y)^{\mathrm{T}} = XE^{\mathrm{T}}$$

从上式可以看出，输入向量 X 越大，则权值的变化Δw 就越大。当存在奇异样本（即该样本向量同其他所有的样本向量比较起来，特别大或者特别小）时，利用以上规则训练时间大为加长。因为其他样本需花很多时间才能同奇异样本所对应的权值变化相匹配。为了消除学习训练时间对奇异样本的敏感性，提出了一种改进的感知机学习规则，也成为标准化感知机学习规则。标准化感知机学习规则试图使奇异样本和其他样本对权值的变化值的影响均衡，且可通过下式实现，即

$$\Delta w = \frac{X}{\|X\|}(T-Y)^{\mathrm{T}} = \frac{X}{\|X\|}E^{\mathrm{T}}$$

标准化感知机的学习函数为 learnpn( )，其调用格式为：

[dW, db]=learnpn(X,E)

式中，X 为输入向量矩阵；E 为误差向量（E=T−Y）；T 表示网络的目标向量；Y 表示网络的输出向量；dW 为权值变化阵；db 为偏值变化阵。

相应于标准化感知机学习规则的训练函数为 trainpn( )。其调用格式为

[W,B,epochs,errors]= trainpn(w,b,X,T,tp)

例 2-4　利用 trainpn( )函数训练一个感知机网络，观察奇异输入样本对训练结果的影响。

解：根据神经网络工具箱函数编写的程序如下：

```
X=[-0.5 -0.5 0.3 -0.1 -80;-0.5 0.5 -0.5 1 100];   %提供训练集
T=[1 1 0 0 1];                                    %提供目标值
[W,b]=initp(X,T);                                 %初始化网络
%训练网络,同时绘制输入样本加网络训练后的分类线
[W,b,epochs,errors]=trainpn(W,b,X,T,-1);
figure;ploterr(errors);                           %绘制误差曲线
X1=X;                                             %给定输入
y=simup(X1,W,b)                                %仿真网络，并给出输出值
```

执行以上程序后可得如下结果，如图 2-9 和图 2-10 所示。

```
y =
  1  1  0  0  1
```

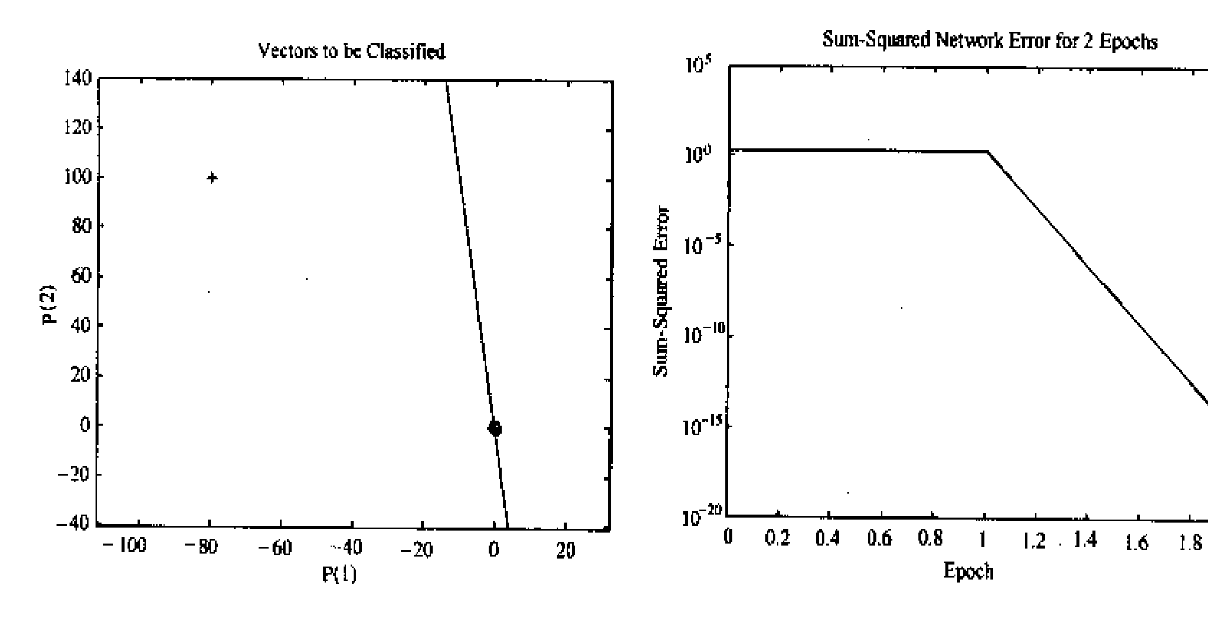


图 2-9　输入样本加网络训练后的分类线　　　图 2-10　误差曲线

由如图 2-10 所示可见，利用函数 trainpn( )，网络训练只需要两步；如果利用函数 trainp( )，则网络训练大多需要经过 60 多步。

11. 建立感知机神经网络函数 newp( )

利用 newp( )函数可建立一个感知机神经网络。其调用格式为

net=newp(Xr,S,Tf,Lf)

式中，Xr 为一个 $r\times2$ 维的输入向量矩阵，它决定了 r 维输入向量的最大值和最小值的取值范围；S 表示神经元的个数；Tf 表示网络的传输函数，默认值为 hardlim；Lf 表示网络的学

习函数，默认值为 learnp；net 为生成的新感知机神经网络。例如，建立一个两输入且样本点取值在[-1，1]之间，而网络只有单个神经元的感知机神经网络，可利用以下命令

```
>>net=newp([-1 1;-1 1],1);
```

使用 plotpc( )函数可以在已绘制的图上加上感知机分类线。让它返回得到的分类线的句柄，以便在下一次再绘制分类线时能够将原来的分类线删除，如

```
>>handle=plotpc(net.iw{1},net.b{1});
```

式中，net.iw{1}用来计算网络 net 的权值，net.b{1}用来计算网络 net 的偏值。

12. 初始化神经网络函数 init( )

利用初始化神经网络函数 init( )可以对一个已存在的神经网络进行初始化修正。该网络的权值和偏值是按照网络初始化函数来进行修正的。其调用格式为

net=init (NET)

式中，NET 为初始化前的网络；net 为初始化后的网络。

13. 神经网络训练函数 train( )

利用 train( )函数可以训练一个神经网络。网络训练函数是一种通用的学习函数，训练函数重复地把一组输入向量应用到一个网络上，每次都更新网络，直到达到了某种准则。停止准则可能是最大的学习步数、最小的误差梯度或误差目标等。其调用格式为

[net,tr]= train(NET,X,T,Pi,Ai)

式中，NET 为要训练的网络；X 为网络的输入向量矩阵；T 表示网络的目标矩阵，默认值为 0；Pi 表示初始输入延时，默认值为 0；Ai 表示初始的层延时，默认值为 0；net 为修正后的网络；tr 为训练步数和性能

例 利用 train( )函数训练一个神经网络的 MATLAB 程序如下：

```
X=[-0.5,-0.5,0.3,-0.1,0.2,0,0.6,0.8;
   -0.5,0.5,-0.5,1,0.5,-0.9,0.8,-0.6];
T=[1 1 0 1 1 0 1 0]; net=newp([-1 1;-1 1],1);
net.performFcn='mae';    %平均绝对误差函数
net.trainParam.goal=0.01; %训练目标误差
net.trainParam.epochs=50; %训练步数
net.trainParam.show=1;    %计算步长
net.trainParam.mc=0.95;   %动量常数
[net1,tr]=train(net,X,T);
```

运行以上程序可得到如图 2-11 所示的训练过程误差曲线。由图可见，网络训练只需 5 步。

14. 网络自适应训练函数 adapt( )

另一种通用的学习函数是自适应函数 adapt( )。自适应函数在每一个输入时间阶段更新


网络时仿真网络，且在进行下一个输入的仿真前完成。其调用格式为

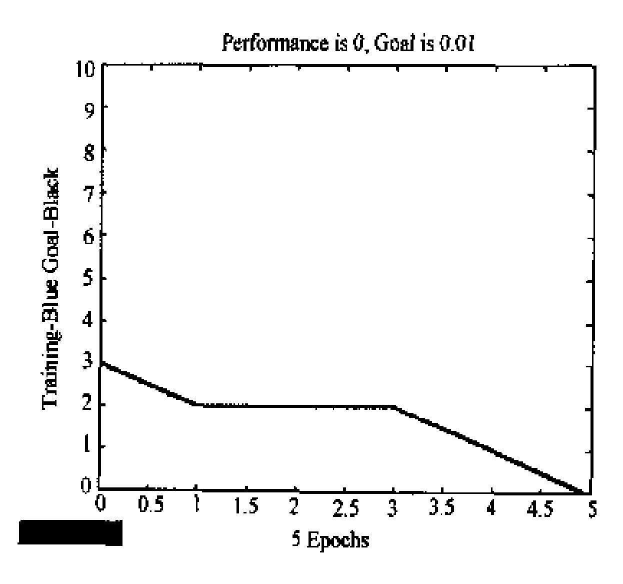


图 2-11　训练过程误差曲线

[net,Y,E,Xf,Af]= adapt(NET,X,T,Xi,Ai)

式中，NET 为要训练的网络；X 为网络的输入向量矩阵；T 表示网络的目标向量，默认值为 0；Xi 表示初始输入延时，默认值为 0；Ai 表示初始的层延时，默认值为 0；net 为修正后的网络；Y 表示网络的输出向量矩阵；E 为网络的误差；Xf 表示最终输入延时；Af 表示最终的层延时。例如以下 MATLAB 程序可训练一个神经网络

```
E=1;
While (mae(E))
 [net,Y,E]=adapt(net,X,T)
 handle=plotpc(net.iw{1},net.b{1},handle);
end
```

以上 m 文件中使用了 mae( )函数得到网络的平均绝对误差，可以此作为一项重要的性能指标函数。

15. 网络仿真函数 sim( )

神经网络一旦训练完成，则网络的权值和偏值就已经确定了，于是就可以使用它来解决实际问题了。利用 sim( )函数可以测试一个神经网络的性能。其调用格式为

[Y,Xf,Af]=sim(net,X,Xi,Ai)

式中，net 为要测试的网络；X 为网络的输入向量矩阵；Xi 表示初始输入延时，默认值为 0；Ai 表示初始的层延时，默认值为 0；Y 表示网络的输出向量矩阵；Xf 表示最终输入延时；Af 表示最终的层延时。例如，选择任意 10 个点进行测试以上网络 net 的 MATLAB 命令如下：

```
>>testX=[-0.5 0.3 -0.9 0.4 -0.1 0.2 -0.6 0.8 0.1 0.4;
```

```
    -0.3 -0.8 -0.4 -0.7 0.4 -0.6 0.1 -0.5 -0.5 0.3]
>>y=sim(net,testX) ;
>>figure;plotpv(testX,y)
>>plotpc(net.iw{1},net.b{1});
```

其测试结果如图 2-12 所示。从测试结果来看，网络能够将它们正确分类，这说明设计的网络是正确的。

例 2-5 利用 train( )函数训练一个感知机网络，使其同样能够完成“或”的功能。

解：根据神经网络工具箱函数编写的程序如下：

```
X=[0 0 1 1;0 1 0 1];T=[0 1 1 1];    %提供四组 2 输入 1 输出的训练集和目标值
net=newp([-1 1;-1 1],1)             %建立网络
[net,Y,E]=train(net,X,T);           %训练网络
X1=X;                               %给定输入
y=sim(net,X1)                       %仿真网络，并给出输出值
```

执行以上程序后可得如下结果及如图 2-13 所示的训练过程误差曲线。

```
y =
    0  1  1  1
```

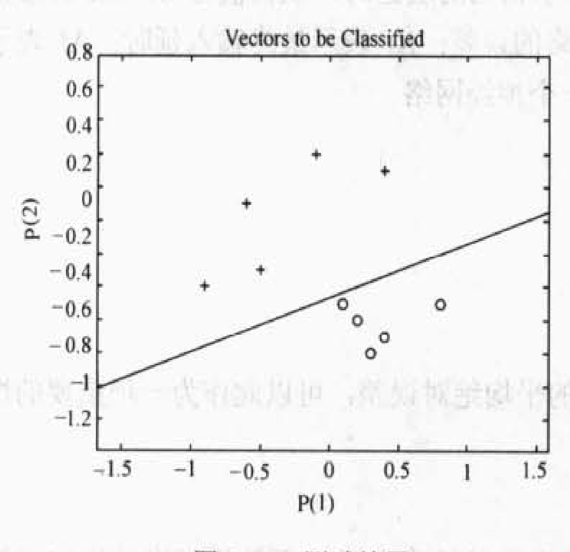


图 2-12 测试结果

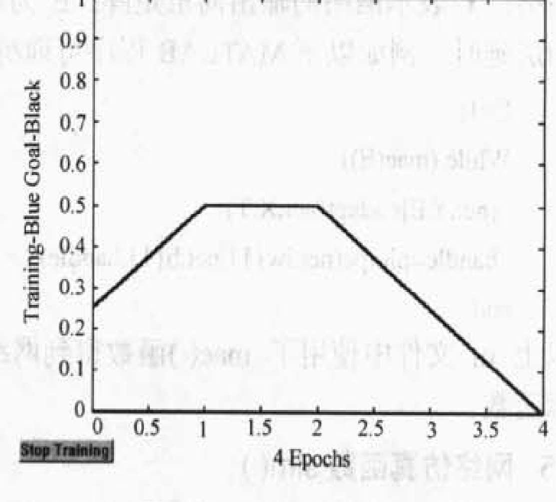


图 2-13 训练过程误差曲线

例 2-6 建立一个感知机网络，使其能够对三个输入进行分类。

解：根据神经网络工具箱函数编写的程序如下：

```
X=[-1 1 -1 1 -1 1 -1 1;-1 -1 1 1 -1 -1 1 1;-1 -1 -1 -1 1 1 1 1];
T=[0 1 0 0 1 1 0 1];                %提供 8 组 3 输入 1 输出的训练集和目标值
plotpv(X,T);                        %绘制输入样本的分类
[W,b]=initp(X,T);                   %初始化网络
```

```
figure;plotpv(X,T);plotpc(W,b);          %绘制输入样本加网络初始分类线
%训练网络,同时绘制输入样本加网络训练后的分类线
figure;
[W,b,epochs,errors]=trainp(W,b,X,T,-1);
figure;ploterr(errors);                   %绘制误差曲线
X1=X;                                     %给定输入
y=simup(X1,W,b)                           %仿真网络，并给出输出值
```

执行以上程序后可得到如下结果，如图 2-14～图 2-17 所示。

```
y =
    0  1  0  0  1  1  0  1
```

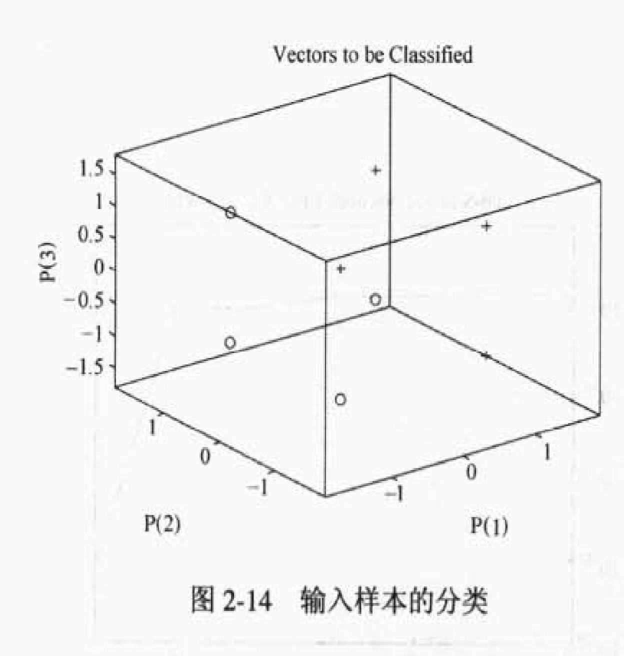


图 2-14　输入样本的分类

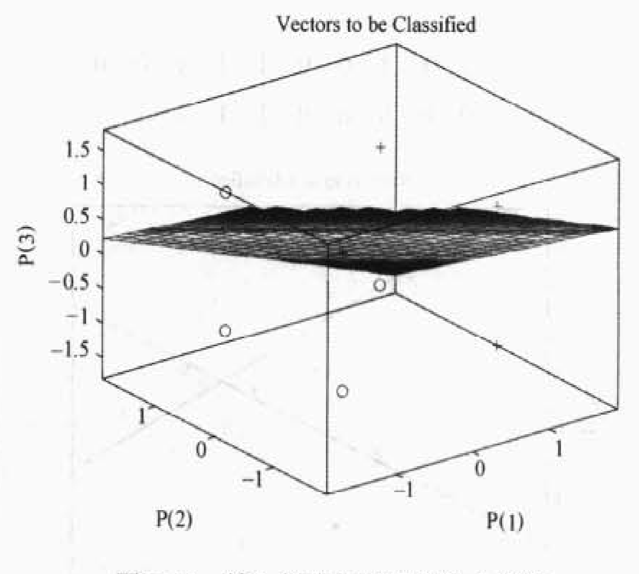


图 2-15　输入样本加网络初始分类线

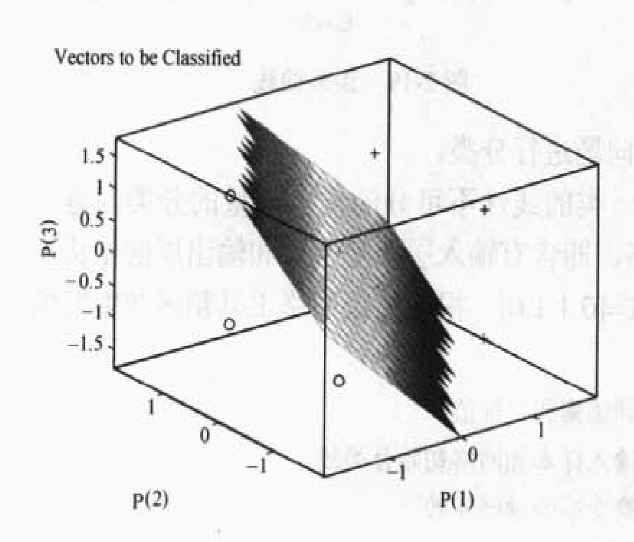


图 2-16　输入样本加网络训练后的分类线

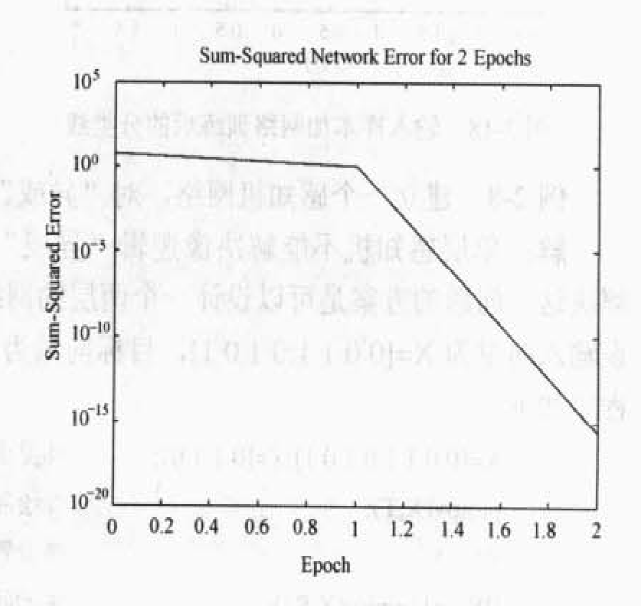


图 2-17　误差曲线

例 2-7 建立一个感知机网络，使其能够将输入分为 4 类。

解：根据神经网络工具箱函数编写的程序如下：

```
X=[0.1 0.7 0.8 0.8 1 0.3 0 -0.3 -0.5 -1.5;1.2 1.8 1.6 0.6 0.8 0.5 0.2 0.8 -1.5 -1.3];
T=[1 1 1 0 0 1 1 1 0 0;0 0 0 0 0 1 1 1 1 1];   %提供训练集和目标值
[W,b]=initp(X,T);                             %初始化网络
%训练网络,同时绘制输入样本加网络训练后的分类线
[W,b,epochs,errors]=trainp(W,b,X,T,-1);
figure;ploterr(errors);                       %绘制误差曲线
X1=X;y=simup(X1,W,b)                          %仿真网络，并给出输出值
```

执行以上程序后可得到如下结果，如图 2-18 和图 2-19 所示。

```
y =
  1  1  1  0  0  1  1  1  0  0
  0  0  0  0  0  1  1  1  1  1
```

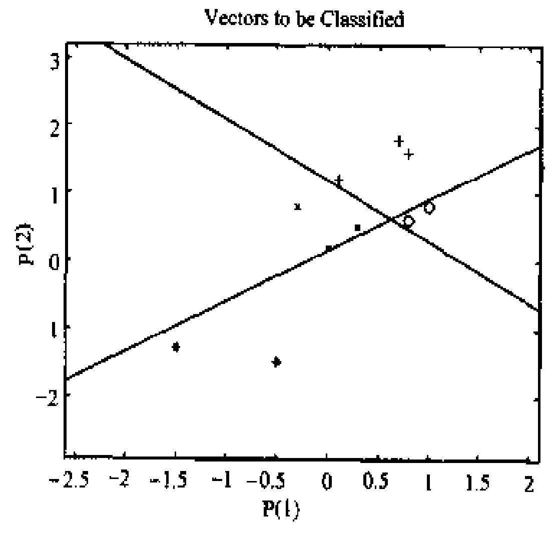



图 2-18 输入样本加网络训练后的分类线

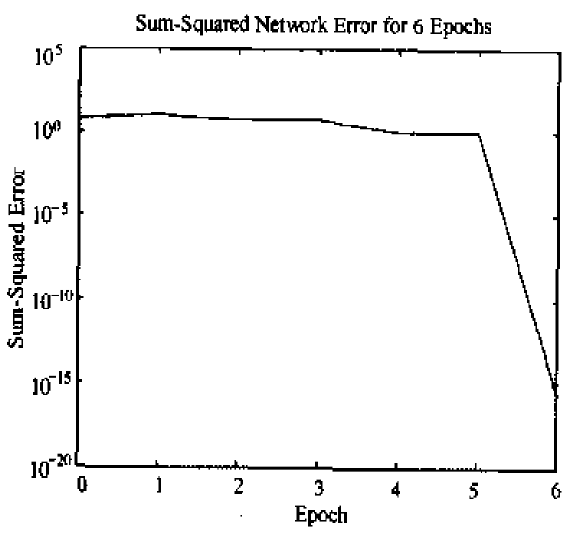


图 2-19 误差曲线

例 2-8 建立一个感知机网络，对“异或”问题进行分类。

解：单层感知机不能解决像逻辑“异或”一类的线性不可分的输入向量的分类问题。解决这一问题的方案是可以设计一个两层的网络，即含有输入层、隐含层和输出层的结构。设输入向量为 X=[0 0 1 1;0 1 0 1]，目标向量为 T=[0 1 1 0]。根据神经网络工具箱函数编写的程序如下：

```
X=[0 0 1 1;0 1 0 1];T=[0 1 1 0];     %提供训练集和目标值
plotpv(X,T);                         %绘制输入样本加网络初始分类线
S1=15;                               %设置隐含层的神经元数
[W1,b1]=initp(X,S1);                 %初始化隐含层
```

```
[W2,b2]=initp(S1,T);                        %初始化输出层
X1=simup(X,W1,b1);                          %计算隐含层输出值，把它作为输出层的输入
%训练输出层的权值和偏值,同时绘制输入样本加网络训练后的分类线
figure;
[W2,b2,epochs,errors]=trainpn(W2,b2,X1,T,-1);
figure;ploterr(errors);                     %绘制误差曲线
X1=X;                                       %给定输入
y1=simup(X1,W1,b1);                         %计算隐含层输出值
y=simup(y1,W2,b2)                           %计算输出层输出值
```

执行以上程序后可得到如下结果，如图 2-20 和图 2-21 所示。

```
y =
    0    1    1    0
```

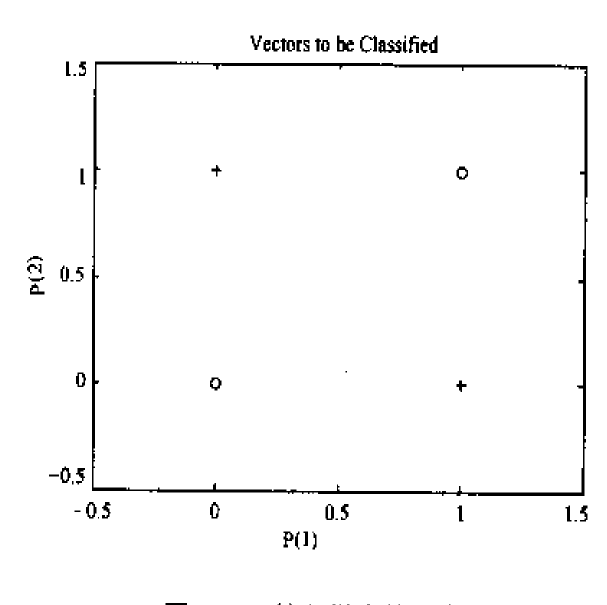


图 2-20　输入样本的分类

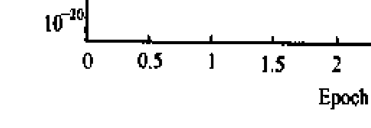


图 2-21　误差曲线

由如图 2-21 所示可见，网络训练只需要 4 步。需要指出的是，由于隐含层的权值和偏值是随机给定的而且不可调整，故隐含层的输出也是随机的。这样网络有可能有解，也有可能无解。如果网络找不到解，则可再次运行网络，以重新初始化隐含层的权值和偏值。如果采用单层网络，则对以上问题永远也找不到正确的分类方案。

2.2　线性神经网络工具箱函数

MATLAB 神经网络工具箱中提供了大量的与线性网络相关的的工具箱函数。在 MATLAB 工作空间的命令行中输入“help linnet”，便可得到与线性网络相关的函数，进一

步利用 help 命令又能得到相关函数的详细介绍。表 2-2 列出了线性网络的重要函数和基本功能。

表 2-2 线性网络的重要函数和基本功能

函数名	功能
sse( )	误差平方和性能函数
purelin( )	线性传输函数
initlin( )	线性神经的初始化函数
solvelin( )	设计一个线性神经网络
simulin( )	对线性神经网络进行仿真
maxlinlr( )	计算线性层的最大学习速率
learnwh( )	Widrow-Hoff 的学习函数
trainwh( )	对线性神经网络进行离线训练
adaptwh( )	对线性神经网络进行在线自适应训练
newlind( )	设计一个线性层
newlin( )	新建一个线性层
dotprod( )	权值点积函数
netprod( )	网络输入的积函数
normprod( )	规范点积权值函数
initwb( )	神经网络某一层的权值和偏值初始化函数
trainwb( )	神经网络的权值和偏值训练函数
adaptwb( )	神经网络的权值和偏值自适应函数
initlay( )	神经网络某一层的初始化函数
initzero( )	将权值设置为零的初始化函数

1. 误差平方和性能函数 sse( )

线性网络学习规则为调整网络的权值和偏值，使网络误差平方和最小。误差平方和性能函数的调用格式为

perf=sse(E,w,pp)

式中，E 为误差矩阵或向量（E=T−Y）；T 表示网络的目标向量；Y 表示网络的输出向量；w 为所有权值和偏值向量（可忽略）；pp 为性能参数（可忽略）；perf 为误差平方和。

2. 线性传输函数 purelin( )

神经元最简单的传输函数是简单地从神经元输入到输出的线性传输函数，输出仅仅被神经元所附加的偏差所修正。线性传输函数常用于由 Widrow-Hoff 或 BP 准则来训练的神经网络中。该函数的调用格式为

a=purelin(N)

或 a=purelin(Z,b)

a=purelin(P)

式中，函数 purelin (N) 为返回网络输入向量 N 的输出矩阵 a，一般情况下，输出矩阵 a 可直接用网络输入向量 N 代替，即输出 a 等于输入 N；函数 purelin(Z,b)用于矢量是成批处理且偏差存在的情况下，此时的偏差 b 和加权输入矩阵 Z 是分开传输的，偏差矢量 b 加到 Z 中的每个矢量中形成网络输入矩阵，最后被返回；函数 purelin(P)包含传输函数的特性名并返回问题中的特性。如下的特性可从任何传输函数中获得：

（1）delta——与传输函数相关的 delta 函数；

（2）init——传输函数的标准初始化函数；

（3）name——传输函数的全称；

（4）output——包含有传输函数最小、最大值的二元矢量。

例如，利用以下命令可得到如图 2-22 所示的线性传输函数。

```
>>n=-5:0.1:5;b=0;
>>a=purelin(n,b);plot(n,a)
```

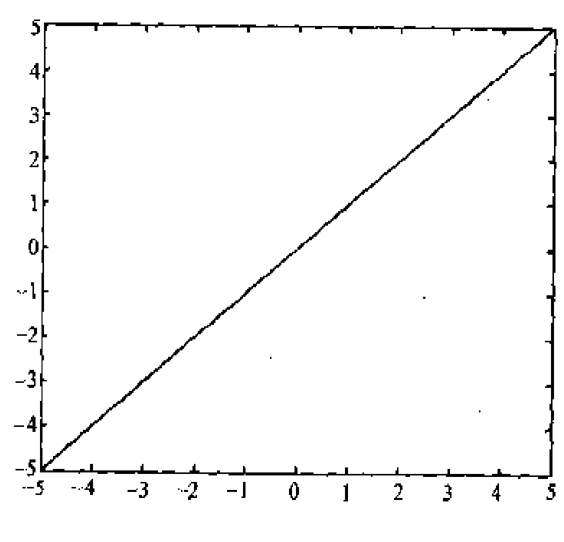


图 2-22　线性传输函数

3. 线性神经网络的初始化函数 initlin( )

利用 initlin( )函数可建立一个单层（一个输入层和一个输出层）线性神经网络。其调用格式为

[W,b]=initlin(R,S)

或

[W,b]=initlin(X,T)

式中，R 为输入个数；S 为输出神经元数；W 为网络的初始权值；b 为网络的初始偏值。另外，R 和 S 可以用对应的输入向量矩阵 X 和目标向量 T 来代替，此时输入个数和输出神经元数根据 X 和 T 中的行数来设置，如以下 MATLAB 命令

```
>>X=[0 0 1 1;0 1 0 1];T=[0 1 1 1]; [W,b]=initlin(X,T);
```

4. 设计一个线性神经网络函数 solvelin( )

与大多数其他神经网络不同的是，只要已知线性神经网络的输入向量和目标向量，就可以直接对其进行设计。使用函数 solvelin( ) 设计的线性神经网络可以不经过训练，直接找出网络的权值和偏值，使得网络的误差的平方和最小。该函数的调用格式为

[W,b]=solvelin(X,T)

式中，X 为 R×Q 维的 Q 组输入向量；T 为 S×Q 维的 Q 组目标分类向量；W 为网络的初始权值；b 为网络的初始偏值。

5. 线性神经网络的仿真函数 simulin( )

利用函数 solvelin( ) 建立的线性神经网络的权值和偏值就已经根据网络的输入向量和目标向量训练好了。simulin( )函数可以测试一个线性神经网络的性能。其调用格式为

Y=simulin(X,w,b)

式中，X 为网络的输入向量矩阵；w 为网络的权值；b 为网络的偏值；Y 表示网络的输出向量。例如，建立一个线性网络可利用以下命令，即

```
>>X=[1 2 3];T=[2.0 4.1 5.9];                %给定训练集和目标值
>>[w,b]=solvelin(X,T);y=simulin(X,w,b)
```

结果显示为

```
y=
     2.0500    4.0000    5.9500
```

6. 计算线性层的最大学习速率函数 maxlinlr( )

函数 maxlinlr( )用于计算用 Widrow-Hoff 准则训练的线性网络的最大稳定学习速率。其调用格式为

lr=maxlinlr(X)

或

lr=maxlinlr(X,b)

式中，X 为输入向量；lr 为学习速率。网络不具有偏值时可采用上式，具有偏值 b 时采用下式。一般地，学习速率越大，网络训练所需的时间越少。但如果太高，则学习就不稳定了。例如，利用以下命令可计算出用 Widrow-Hoff 准则训练的在所给输入下线性神经网络所用的学习率上限。

```
>>X=[1 2 -4 7;0.1 3 10 6]; lr=maxlinlr(X)
```

其结果显示为

```
lr =
     0.0069
```

7. 线性神经网络学习函数 learnwh( )

线性网络采用 Widrow-Hoff 学习规则。Widrow-Hoff 学习规则只能训练单层的线性神经

网络，但这并不影响单层线性神经网络的应用，因为对每一个多层线性神经网络而言，都可以设计出一个性能完全相当的单层线性神经网络。当利用函数 solvelin( )设计的线性神经网络不能调整网络的权值和偏值而使网络的误差平方和性能最小时，则可以应用函数 learnwh( )和函数 trainwh( )来调整网络的权值和偏值。函数 learnwh( )的调用格式为

[dW, db]=learnwh(X,E,lr)

式中，X 为输入向量矩阵；E 为误差向量（E=T−Y）；T 表示网络的目标向量；Y 表示网络的输出向量；dW 为权值变化阵；db 为偏值变化阵；lr 为学习速率（$0<lr\leqslant 1$），用于控制每次误差的修正值。学习速率 lr 较大时，学习过程加速，网络收敛较快；但当 lr 太大时，学习过程则变得不稳定，且误差会加大。因此，学习速率的取值很关键。它可利用函数 maxlinlr( )求出合适的学习速率 lr。

8. 线性神经网络的训练函数 trainwh( )

函数 trainwh( )可利用 Widrow-Hoff 学习规则对线性层的权值进行训练，利用输入矢量计算该层的输出矢量，然后根据产生的误差矢量调整该层的权值和偏差。该函数的调用格式为

[W,B,epochs,errors]= trainwh(w,b,X,T,tp)

式中，w 和 W 分别为网络训练前后的权值；b 和 B 分别为网络训练前后的偏值；X 为网络的输入向量矩阵；T 表示网络的目标向量；tp=[disp_freq　max_epoch　err_goal　lr]是训练控制参数，包括更新显示的迭代次数 disp_freq（默认值为 25）、训练的最大迭代次数 max_epoch（默认值为 100）、目标误差平方和 err_goal（默认值为 0.02）和学习速率 lr（用函数 maxlinlr( )找默认值）；epochs 表示训练步数；errors 表示训练后的网络误差。

9. 线性神经网络自适应训练函数 adaptwh( )

该函数 adaptwh( )可以利用 Widrow-Hoff 学习规则对线性层的权值进行自适应调节，在每一步迭代过程中，修改自适应线性网络层的权值、偏差和输出矢量，从而学习并适应环境的变化。其调用格式为

[Y,E,W,B]= adaptwh(w,b,X,T,lr)

式中，w 和 W 分别为网络自适应训练前后的权值；b 和 B 分别为网络自适应训练前后的偏值；X 为网络的输入向量矩阵；T 表示网络的目标向量；lr 为学习速率；Y 表示网络的输出向量矩阵；E 为网络的误差。

10. 设计一个线性层函数 newlind( )

利用函数 newlind( )设计出的线性网络已经训练好，可直接使用。该函数的调用格式为

net=newlind(X,T)

式中，X 为 R×Q 维的 Q 组输入向量；T 为 S×Q 维的 Q 组目标分类向量；net 为生成的新线性神经网络。例如，建立一个线性网络可利用以下命令，即

```
>>X=[1 2 3];T=[2.0 4.1 5.9];                    %给定训练集和目标值
```

```
>>net=newlind(X,T);y=sim(net,X)
```

其结果显示为

```
y=
        2.0500      4.0000      5.9500
```

11. 新建一个线性层函数 newlin( )

利用函数 newlin( )设计的线性网络还需训练。该函数的调用格式为

net=newlin(Xr,S,Id,lr)

式中，Xr 为一个输入向量，它决定了输入向量的最小值和最大值的取值范围；S 为输出向量的个数；Id 为输出延时向量，默认值为 0；lr 为学习速率，默认值为 0.01；net 为生成的线性神经网络。例如，建立一个线性网络可利用以下命令，即

```
>>X=[1 2 3];S=1;net=newlin(minmax(X),S);
```

12. 权值点积函数 dotprod( )

网络输入向量与权值的点积可得到加权输入。函数 dotprod ( )的调用格式为

Z=dotprod (W,X)

式中，W 为 S×R 维的权值矩阵；X 为 Q 组 R 维的输入向量；Z 为 Q 组 S 维的 W 与 X 的点积。

13. 网络输入的积函数 netprod( )

所谓网络输入的积函数，是根据加权输入和偏值计算一层的网络输出。其调用格式为

Z=netprod(Z1,Z2,...)

式中，Zi 为 S×Q 维矩阵。

14. 神经网络某一层的权值和偏值初始化函数 initwb( )

利用初始化函数 initwb( )可以对一个已存在的神经网络 NET 的某一层 i 的权值和偏值进行初始化修正。该网络的权值和偏值是按照网络初始化函数来进行修正的。其调用格式为：

net=initwb(NET,i)

式中，NET 为初始化前的网络；i 为第 i 层；net 为第 i 层的权值和偏值修正后的网络。

15. 神经网络某一层的初始化函数 initlay( )

利用初始化函数 initlay( ) 可以对一个已存在的神经网络 NET 的某一层进行初始化修正。该网络的权值和偏值是按照网络初始化函数来进行修正的。其调用格式为

net=initlay (NET)

式中，NET 为初始化前的网络；net 为某一层初始化后的网络。

例 2-9 利用线性网络进行系统辨识。

解：根据以上命令利用 MATLAB 编写的程序如下：

```
%定义输入信号
```

```
Time=0:0.025:5;X=sin(sin(Time.*Time*10));
%定义系统变换函数 y=kx+b，并绘制系统输入 X 与输出 T 曲线
T=2*X+0.8;
figure;plot(Time,X ,Time,T,'--');
%将信号 T 延时 0~2 个时间步长得到网络的输入 X1
Q=length(T); X1=zeros(3,Q); X1(1,1:Q)=T(1,1:Q);
X1(2,2:Q)=T(1,1:(Q-1)); X1(3,3:Q)=T(1,1:(Q-2));
[w,b]=solvelin(X1,T);                %设计网络
y=simulin(X1,w,b);                   %仿真网络
%绘制网络预测输出 y 与系统输出 T 及其误差
figure;plot(Time,y, '+',Time,T,'-' ,Time,T-y,'x');
```

其执行结果如图 2-23 和图 2-24 所示。

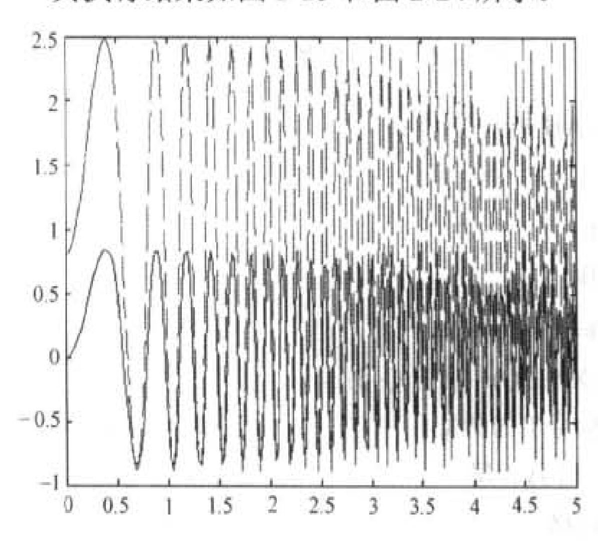


图 2-23 系统输入/输出曲线

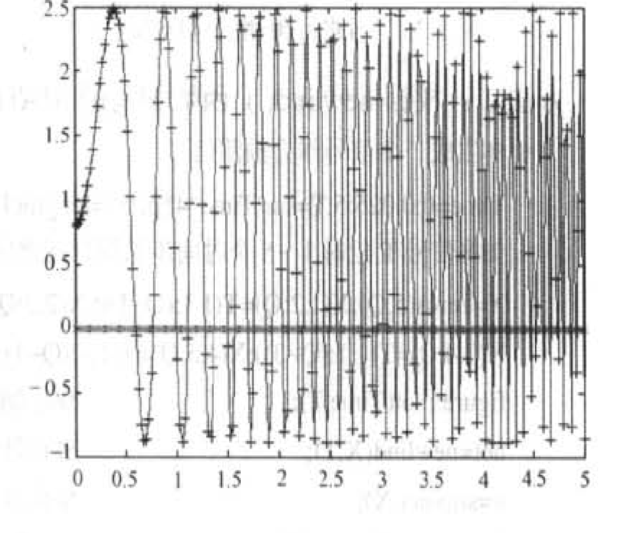


图 2-24 网络预测输出与系统输出和误差

例 2-10 利用线性网络进行自适应预测。

解：方法一：根据 adaptwh( )函数编写的 MATLAB 程序如下：

```
%生成一个信号作为预测信号
Time=0:0.025:5;T=sin(Time*4*pi);Q=length(T);
%将信号 T 延时 1~5 个时间步长得到网络的输入 X
X=zeros(5,Q);X(1,2:Q)=T(1,1:(Q-1)); X(2,3:Q)=T(1,1:(Q-2));
X(3,4:Q)=T(1,1:(Q-3));X(4,5:Q)=T(1,1:(Q-4));X(5,6:Q)=T(1,1:(Q-5));
figure;plot(Time,T);                 %绘制信号 T 的曲线
[W,b]=initlin(X,T);                  %初始化一个线性层
lr=0.1;                              %学习率
[y,E,W,b]= adaptwh(W,b,X,T,lr)       %仿真网络
```

```
figure;plot(Time,y,Time,T,'o');                %绘制网络预测输出及其目标值
```

其执行结果如图 2-25 和图 2-26 所示。

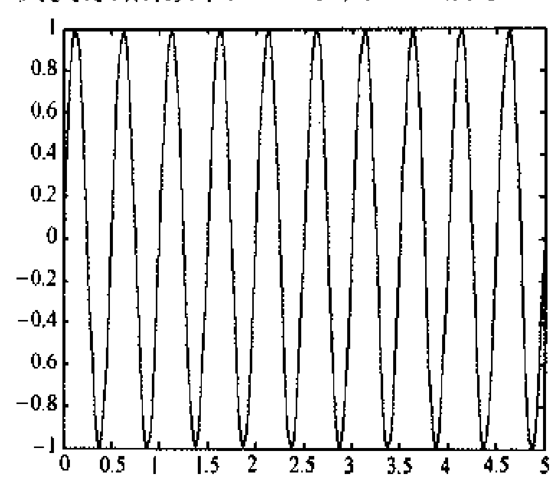


图 2-25 网络待预测的目标信号

图 2-26 网络预测输出与目标值

方法二：根据 newlind( ) 函数编写的 MATLAB 程序如下：

```
%生成一个信号作为预测信号
Time=0:0.025:5;T=sin(Time*4*pi);Q=length(T);
%将信号 T 延时 1~5 个时间步长得到网络的输入 X
X=zeros(5,Q);X(1,2:Q)=T(1,1:(Q-1)); X(2,3:Q)=T(1,1:(Q-2));
X(3,4:Q)=T(1,1:(Q-3));X(4,5:Q)=T(1,1:(Q-4));X(5,6:Q)=T(1,1:(Q-5));
figure;plot(Time,T);                 %绘制信号 T 的曲线
net=newlind(X,T);                    %设计一个线性层
y=sim(net,X);                        %仿真网络
figure;plot(Time,y, Time,T-y,'o');   %绘制网络预测输出及其误差
```

其执行结果如图 2-27 和图 2-28 所示。

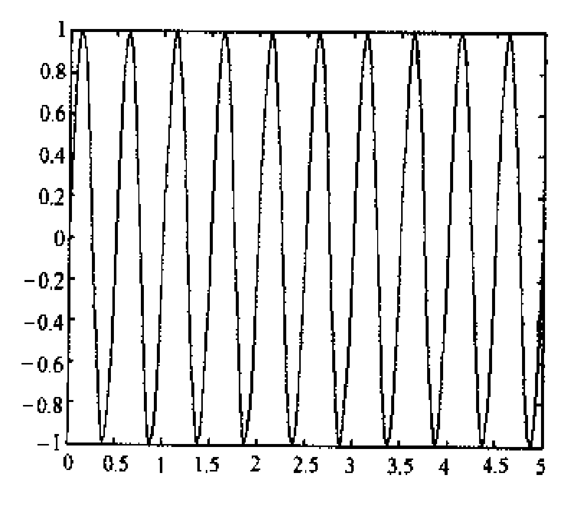


图 2-27 网络待预测的目标信号

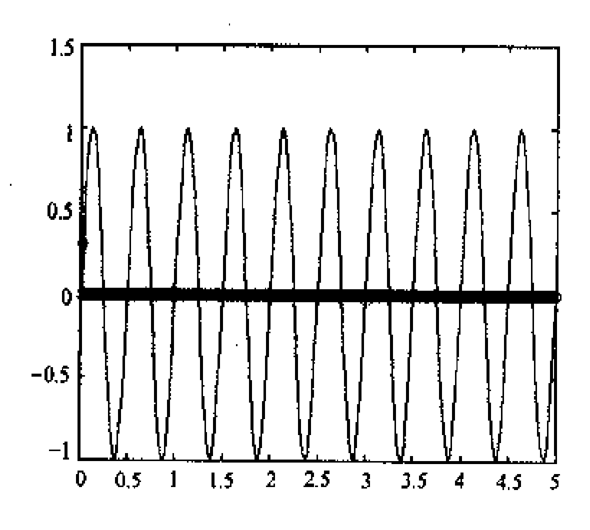


图 2-28 网络预测输出与误差

例 2-11　利用线性网络预测一个时变信号序列。

解：根据 newlin( ) 函数编写的 MATLAB 程序如下：

```
%分别定义两段时间，对应信号的不同频率时段
Time1=0:0.05:4;Time2=4.05:0.024:6;Time=[Time1 Time2];
%获得待预测的目标信号，并绘制网络待预测的目标信号
T=[cos(Time1*4*pi) cos(Time2*8*pi)];plot(Time,T);X=T;
Id=[1 2 3 4 5]; lr=0.1;net=newlin(minmax(X),1,Id,lr);    %新建一个线性层
[net,e,y]=adapt(net,X,T);                                %对网络进行自适应训练
%绘制网络预测输出 y、目标信号 T 及其误差信号 e
figure;plot(Time,y,Time,X,'x',Time,e,'o');
```

其执行结果如图 2-29 和图 2-30 所示。

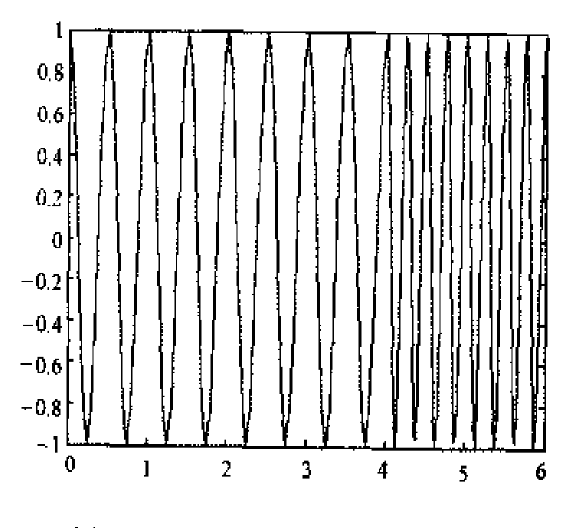


图 2-29　网络待预测的目标信号

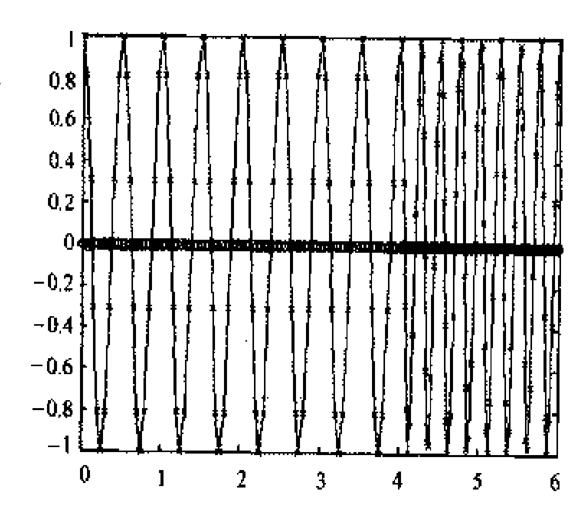


图 2-30　网络预测输出目标信号与误差

2.3　BP 神经网络工具箱函数

MATLAB 神经网络工具箱提供了大量的进行 BP 网络分析和设计的工具箱函数。在 MATLAB 工作空间的命令行输入“help backprop”，便可得到与 BP 神经网络相关的函数，进一步利用 help 命令又能得到相关函数的详细介绍。表 2-3 列出了 BP 神经网络的重要函数和基本功能。

表 2-3　BP 神经网络的重要函数和基本功能

函 数 名	功　能
tansig( )	双曲正切 S 型(Tan-Sigmoid)传输函数
purelin( )	线性(Purelin)传输函数
logsig( )	对数 S 型(Log-Sigmoid)传输函数

续表

函 数 名	功 能
deltatan( )	Tansig 神经元的 delta 函数
deltalin( )	Purelin 神经元的 delta 函数
deltalog( )	Logsig 神经元的 delta 函数
learnbp( )	BP 学习规则
learnbpm( )	含动量规则的快速 BP 学习规则
learnlm( )	Levenberg-Marguardt 学习规则
initff( )	对 BP 神经网络进行初始化
trainbp( )	利用 BP 算法训练前向网络
trainbpx( )	利用快速 BP 算法训练前向网络
trainlm( )	利用 Levenberg-Marguardt 规则训练前向网络
simuff( )	BP 神经网络进行仿真
newff( )	生成一个前馈 BP 网络
newfftd( )	生成一个前馈输入延时 BP 网络
newcf( )	生成一个前向层叠 BP 网络
nwlog( )	对 Logsig 神经元产生 Nguyen-Midrow 随机数
mse( )	均方误差性能函数
sumsqr( )	计算误差平方和
errsurf( )	计算误差曲面
plotes( )	绘制误差曲面图
plotep( )	在误差曲面图上绘制权值和偏值的位置
ploterr( )	绘制误差平方和对训练次数的曲线
barerr( )	绘制误差的直方图

1. 双曲正切S型（Sigmoid）传输函数tansig( )

双曲正切 Sigmoid 函数把神经元的输入范围从(−∞,+∞)映射到(−1,+1)。它是可导函数，适用于BP训练的神经元。该函数的调用格式为

a=tansig(N)

或

a=tansig(Z,b)

a=tansig (P)

其中，函数 tansig (N)为返回网络输入向量 N 的输出矩阵 a；函数 tansig (Z,b)用于矢量是成批处理且存在偏差的情况下，此时的偏差 b 和加权输入矩阵 Z 是分开传输的，偏差矢量 b 加到 Z 中的每个矢量可形成网络输入矩阵，最后被返回；函数 tansig (P)包含传输函数的特性名并返回问题中的特性。如下的特性可从任何传输函数中获得：

（1）delta——与传输函数相关的 delta 函数；

（2）init——传输函数的标准初始化函数；

（3）name——传输函数的全称；

（4）output——包含有传输函数最小、最大值的二元矢量。

例如，利用以下命令可得如图 2-31 所示的双曲正切曲线。

```
>>n=-5:0.1:5;b=0;a=tansig(n,b);plot(n,a)
```

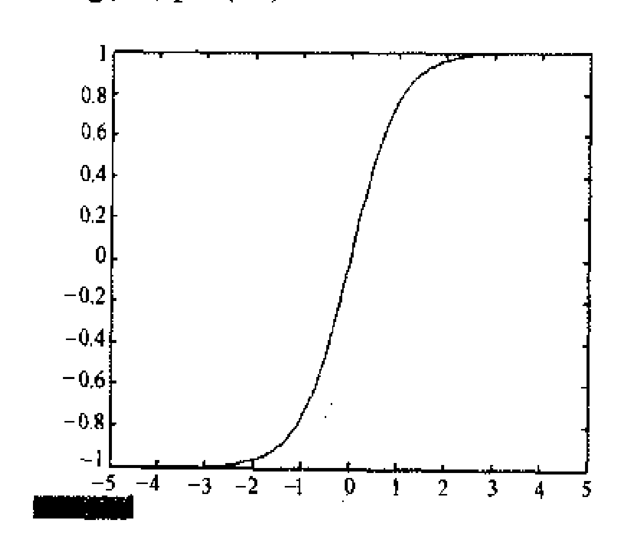


图 2-31　双曲正切曲线

函数 purelin( )和 logsig( )的用法同上。如果 BP 网络的最后一层是 Sigmoid 型神经元，那么整个网络的输出就被限制在一个较小的范围内；如果 BP 网络的最后一层是 Purelin 型线性神经元，那么整个网络的输出可以取任意值。

2. 正切 S 型（Tansig）神经元的求导函数 deltatan ( )

反向传播误差算法(BP)是利用误差平方和对网络各层输入的导数来调整其权值和偏值的，从而降低误差平方和。从网络误差矢量中可推导出输出层的误差导数或δ(delta)矢量，以及隐含层的δ矢量的导出。这种δ矢量的反向传播正是 BP 算法的由来。该函数的调用格式为

dy=deltatan(y)

或　dy=deltatan(y,e)

dy=deltatan(y,d2,w2)

其中，deltatan(y)可计算出这一层输出 y 对本层输出的导数 dy；deltatan(y,e)可计算出 Tansig 输出层的误差导数 dy，参数 y 和 e 分别为该层的输出向量和误差；deltatan(y,d2,w2)可计算出 Tansig 隐含层的误差导数 dy，参数 y 为正切 S 型层的输出向量，d2 和 w2 为下一层的δ(delta)矢量和连接权值。

3. 线性（Purelin）神经元的求导函数 deltalin( )

该函数的调用格式为

dy=deltalin(y)

或 dy=deltalin(y,e)

dy=deltalin(y,d2,w2)

其中，deltalin(y)可计算出这一线性层输出 y 对本层输出的导数 dy；deltalin(y,e) 可计算出线性输出层的误差导数 dy，参数 y 和 e 分别为该层的输出向量和误差；deltalin(y,d2,w2)可计算出 Purelin 隐含层的误差导数 dy，参数 y 为线性层的输出向量，d2 和 w2 为下一层的 delta 矢量和连接权值。

Logsig 神经元的求导函数 deltalog( ) 的用法同上。

4. BP 学习规则函数 learnbp( )

BP 神经网络学习规则为调整网络的权值和偏值使网络误差的平方和为最小。这是通过在梯度下降最陡方向上不断地调整网络的权值和偏值来达到的，计算网络输出层的误差矢量导数，然后反馈回网络，直到每个隐含层的误差导数（称为 delta）都达到要求。这可由函数 deltatan( ) 、deltalin( ) 和 deltalog( ) 计算。根据 BP 准则，每一层的权值矩阵 w 利用本层的δ向量和输入向量 x 来更新，即$\Delta w(i,j)=\eta\delta(i)x(j)$。该函数的调用格式为

[dW,dB]=learnbp(X,delta,lr)

式中，X 为本层的输入向量；delta 为误差导数δ矢量；lr 为学习速率；dW 为权值修正阵；dB 为偏值修正向量。

5. 含动量规则的 BP 学习规则函数 learnbpm( )

为了提高 BP 算法的学习速度并增加算法的可靠性，在 BP 学习算法中引进了动量因子。它使权值的变化等于上次权值的变化与这次由 BP 准则引起的变化之和，这样可将动量加到 BP 学习中，上一次权值变化的影响可由动量常数来调整。动量法降低了网络对于误差曲面局部细节的敏感性，有效地抑制网络陷于局部极小。而自适应学习率也可以使训练时间大大缩短。当动量常数为 0 时，说明权值的变化仅由梯度决定；当动量常数为 1 时，说明新的权值变化仅等于上次权值变化，而忽略掉梯度项。其数学表达式为$\Delta w(i,j)=D\Delta w(i,j)+(1-D)\eta\delta(i)x(j)$。该函数的调用格式为

[dW,dB]=learnbpm(X,delta,lr,D,dw,db)

式中，X 为本层的输入向量；delta 为误差导数δ矢量；lr 为自适应学习速率；D 为动量常数；dw 为上一次权值修正阵；db 为上一次偏值修正向量；dW 为本次权值修正阵；dB 为本次偏值修正向量。

6. Levenberg-Marguardt 学习规则函数 learnlm( )

函数 learnlm( )采用了 Levenberg-Marguardt 优化方法，从而使得学习时间更短。其缺点是，对于复杂的问题，该方法需要很大的存储空间。LM 方法更新参数（如权值和偏值）的数学表达式为$\Delta w=(J^TJ+\mu I)^{-1}J_e^T$。随着μ的增大，LM 方法中的 J^TJ 项变得无关紧要，因而学习过程由$\mu^{-1}J_e^T$决定，即梯度下降法。该函数调用的格式为

[dW,dB]=learnlm(X,delta)

式中，X 为本层的输入向量；delta 为误差导数δ矢量；dW 为权值修正阵；dB 为偏值修正向量。

例 2-12 利用两层 BP 神经网络训练加权系数。两组 3 输入为[1 2;-1 1;1 3]，希望的输出均为[1,1]。隐含层的激活函数取 S 型传输函数，输出层的激活函数取线性传输函数。

解：根据 BP 学习算法的计算步骤，利用 MATLAB 的神经网络工具箱的有关函数编写的程序如下：

```
%两层 BP 算法的第一阶段学习期（训练加权系数 Wki,Wij）
%初始化
lr=0.05;                          %lr 为学习速率
err_goal=0.001;                   % err_goal 为期望误差最小值
max_epoch=10000;                  %训练的最大次数
X=[1 2;-1 1;1 3];T=[1 1;1 1];     %提供两组 3 输入 2 输出的训练集和目标值
%初始化 Wki,Wij(M—输入节点 j 的数量,q—隐含层节点 i 的数量,L—输出节点 k 的数量)
[M,N]=size(X);q=10;[L,N]=size(T); %N-为训练集对数量
Wij=rand(q,M);                    %随机给定输入层与隐含层间的权值
Wki=rand(L,q);                    %随机给定隐含层与输出层间的权值
b1=zeros(q,1);b2=zeros(L,1);      %随机给定隐含层、输出层的偏值
for epoch=1:max_epoch
    Oi=tansig(Wij*X,b1);          %计算网络隐含层的各神经元输出
    Ok=purelin(Wki*Oi,b2);        %计算网络输出层的各神经元输出
    E=T-Ok;                       %计算网络误差
    deltak=deltalin(Ok,E);        %计算输出层的 delta
    deltai=deltatan(Oi,deltak,Wki); %计算隐含层的 delta
    %调整输出层加权系数
    [dWki,db2]=learnbp(Oi,deltak,lr);
    Wki=Wki+dWki;b2=b2+db2;
    %调整隐含层加权系数
    [dWij,db1]=learnbp(X,deltai,lr);
    Wij=Wij+dWij;b1=b1+db1;
    %计算网络权值修正后的误差平方和
    SSE=sumsqr(T-purelin(Wki*tansig(Wij*X,b1),b2));
    if (SSE<err_goal) break;end
end
epoch                             %显示计算次数
%Wij,Wki                          %显示加权系数
```

```
%BP算法的第二阶段工作期（根据训练好的Wki,Wij和给定的输入计算输出）
X1=X;                          %给定输入
Oi=tansig(Wij*X1,b1);          %计算网络隐含层的各神经元输出
Ok=purelin(Wki*Oi,b2);         %计算网络输出层的各神经元输出
Ok                             %显示网络输出层的输出
```

其结果显示为

```
Ok =
    1.0068    0.9971
    0.9758    1.0178
```

7. BP神经网络初始化函数 initff( )

在设计一个BP网络时，只要已知网络的输入向量的取值范围、各层的神经元个数及传输函数，就可以利用初始化函数 initff( ) 对BP神经网络进行初始化。函数 initff( ) 可最多对三层神经网络进行初始化即可得到每层的权值和偏值。其调用格式为

[W,B]=initff (Xr,S,'Tf')

或 [W1,B1,W2,B2]=initff (Xr,S1, 'Tf1', S2, 'Tf2')

[W1,B1,W2,B2,W3,B3]=initff (Xr,S1, 'Tf1', S2, 'Tf2', S3, 'Tf3')

式中，Xr 为一个向量矩阵，它决定了输入向量的最小值和最大值的取值范围；S,Si 为各层神经元的个数；Tf ,Tfi 为各层的传输函数；W 和 Wi 为初始化后各层的权值矩阵；B 和 Bi 为初始化后各层的偏值向量。另外，输出层神经元的个数 S 或 Si 可以用对应的目标向量 T 来代替，此时输出神经元数根据 T 中的行数来设置。

例如，设计一个隐含层有8个神经元、传输函数为 tansig、输出层有5个神经元和传输函数为 purelin 的两层BP神经网络可利用以下命令，即

```
>>X=[sin(0:100);cos([0:100]*2)];
>>[W1,B1,W2,B2]=initff(X,8,'tansig',5,'purelin')
```

8. 利用BP算法训练前向网络函数 trainbp( )

BP神经网络学习规则为调整网络的权值和偏值使网络误差的平方和为最小。这是通过在梯度下降方向上不断地调整网络的权值和偏值来达到的。该函数的调用格式为

[W,B,te,tr]=trainbp(w,b, 'Tf',X,T,tp)

或 [W1,B1,W2,B2,te,tr]=trainbp(w1,b1, 'Tf1',w2 ,b2, 'Tf2' , X,T,tp)

[W1,B1,W2,B2, W3,B3,te,tr]=trainbp(w1,b1, 'Tf1',w2 ,b2, 'Tf2' ,w3,b3, 'Tf3',X,T,tp)

式中，w 和 W(及 wi 和 Wi)分别为训练前后的权值矩阵；b 和 B(及 bi 和 Bi)分别为训练前后的偏值向量；te 为实际训练次数；tr 为网络训练误差平方和的行向量；Tf 和 Tfi 为传输函数；X 为输入向量；T 为目标向量；tp=[disp_freq max_epoch err_goal lr]是训练控制参数，其作用是设定如何进行训练，其中 tp(1)显示间隔次数，默认值为 25；tp(2)显示最大循

环次数，默认值为 100；tp(3)为目标误差平方和，默认值为 0.02；tp(4)为学习速率，默认值为 0.01。

9. 利用快速 BP 算法训练前向网络函数 trainbpx( )

使用动量因子时，BP 算法可找到更好的解，而自适应学习率也可以使训练时间大大缩短。该函数的调用格式为

[W,B,te,tr]=trainbpx(w,b, 'Tf',X,T,tp)

或 [W1,B1,W2,B2,te,tr]=trainbpx(w1,b1,'Tf1',w2,b2,'Tf2',X,T,tp);

[W1,B1,W2,B2, W3,B3,te,tr]=trainbpx(w1,b1, 'Tf1',w2 ,b2, 'Tf2' ,w3,b3, 'Tf3',X,T,tp)

式中，w 和 W(及 wi 和 Wi)分别为训练前后的权值矩阵；b 和 B(及 bi 和 Bi)分别为训练前后的偏值向量；te 为实际训练次数；tr 为网络训练误差平方和的行向量；Tf 和 Tfi 为传输函数；X 为输入向量；T 为目标向量；tp 是训练控制参数，其作用是设定如何进行训练，其中 tp(1)显示间隔次数，默认值为 25；tp(2)为最大循环次数，默认值为 100；tp(3)为目标误差平方和，默认值为 0.02；tp(4)为学习速率，默认值为 0.01；tp(5)为学习速率增长系数，默认值为 1.05；tp(6)学习速率减小系数，默认值为 0.7；tp(7)为动量常数，默认值为 0.9；tp(8)为最大误差率，默认值为 1.04。

10. 利用 Levenberg-Marguardt 规则训练前向网络函数 trainlm( )

函数 trainbp( )和 trainbpx( )都是基于梯度下降的训练算法，而函数 trainblm( )是建立在一种优化方法基础上的训练算法。其调用格式为

[W,B,te,tr]=trainlm(w,b, 'Tf',X,T,tp)

或 [W1,B1,W2,B2,te,tr]=trainlm(w1,b1, 'Tf1',w2 ,b2, 'Tf2' , X,T,tp)

[W1,B1,W2,B2, W3,B3,te,tr]=trainlm(w1,b1, 'Tf1',w2 ,b2, 'Tf2' ,w3,b3, 'Tf3',X,T,tp)

式中，w 和 W(及 wi 和 Wi)分别为训练前后的权值矩阵；b 和 B(及 bi 和 Bi)分别为训练前后的偏值向量；te 为实际训练次数；tr 为网络训练误差平方和的行向量；Tf 和 Tfi 为传输函数；X 为输入向量；T 为目标向量；tp 是训练控制参数，其作用是设定如何进行训练，其中 tp(1)显示间隔次数，默认值为 25；tp(2)为最大循环次数，默认值为 100；tp(3)为目标误差，默认值为 0.02；tp(4)为最小梯度，默认值为 0.001；tp(5)为学习速率η的初始值，默认值为 0.001；tp(6)为参数η的增加系数，默认值为 10；tp(7)为参数η的减小系数，默认值为 0.1；tp(8)为参数η的最大值，默认值为 10。

函数 trainlm( )的训练速度最快，但它需要更大的存储空间，trainbpx( )的训练速度次之，trainbp( )最慢。

11. BP 神经网络仿真函数 simuff( )

BP 神经网络由一系列网络层组成。每一层都从前一层得到输入数据。simuff( )函数可仿真最多三层前向网络。其调用格式为

y=simuff (X,w,b,'Tf',)

或 [y1,y2]=simuff (X,w1,b1,'Tf1',w2,b2,'Tf2')

[y1,y2,y3]=simuff (X,w1,b1,'Tf1',w2,b2,'Tf2',,w3,b3,'Tf3',)

式中，X 为输入向量；w,wi 为权值矩阵；b,bi 为偏值矩阵；Tf,Tfi 为传输函数；y,yi 为各层的输出向量矩阵。

例 2-13 利用两层 BP 神经网络完成函数逼近。隐含层的激活函数取 S 型传输函数，输出层的激活函数取线性传输函数。

解：（1）根据神经网络工具箱函数 trainbp( ) 编写的程序如下：

```
X=-1:0.1:1;T=sin(pi*X);                          %提供训练集和目标值
plot(X,T,'+');
%建立网络，并得权值和偏值
[R,N]=size(X);[S2,N]=size(T);S1=5;
[W1,b1,W2,b2]=initff(X,S1,'tansig',S2,'purelin');
[y1,y21]=simuff(X,W1,b1,'tansig',W2,b2,'purelin');
%利用不含噪声的理想输入数据训练网络
disp_freq=10;                                    %显示间隔
max_epoch=8000;                                  %训练时间
err_goal=0.02;                                   %训练目标误差
lr=0.01;                                         %学习速率
tp=[disp_freq   max_epoch   err_goal lr];
[W1,b1,W2,b2,te,tr]=trainbp(W1,b1,'tansig',W2,b2,'purelin',X,T,tp);
[y1,y22]=simuff(X,W1,b1,'tansig',W2,b2,'purelin');
plot(X,T,'-',X,y21,'--',X,y22,'o')
X1=0.5; y2=simuff(X1,W1,b1,'tansig',W2,b2,'purelin')
```

利用以上程序可得如图 2-32 所示的训练前后输出与目标值及如下结果，即

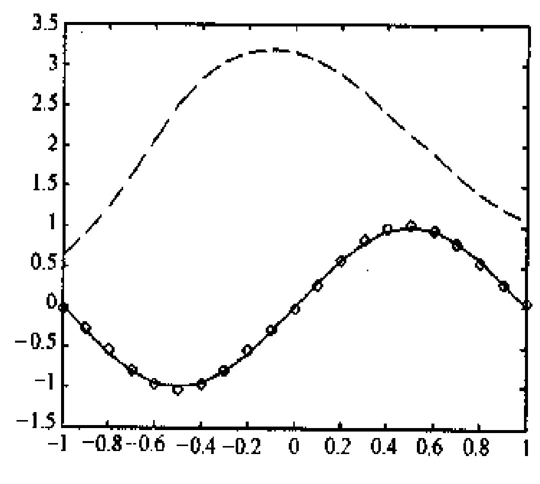


图 2-32 训练前后输出与目标值

```
y2 =
    0.9887
```

（2）根据神经网络工具箱函数 trainbpx( ) 编写的程序如下：

```
X=-1:0.1:1;T=sin(pi*X);
plot(X,T,'+');
%建立网络，并得权值和偏值
[R,N]=size(X);[S2,N]=size(T);S1=5;
[W1,b1,W2,b2]=initff(X,S1,'tansig',S2,'purelin');
%利用不含噪声的理想输入数据训练网络，并得权值和偏值
disp_freq=10;                          %显示间隔
max_epoch=8000;                        %训练时间
err_goal=0.02;                         %训练目标误差
lr=0.01;                               %学习速率
mc=0.95;                               %动量因子
tp=[disp_freq  max_epoch  err_goal lr NaN NaN mc];
[W1,b1,W2,b2,te,tr]=trainbpx(W1,b1,'tansig',W2,b2,'purelin',X,T,tp);
X1=0.5;
[y1,y2]=simuff(X1,W1,b1,'tansig',W2,b2,'purelin');
y2
```

利用以上程序可得如下结果，即

```
y2 =
    1.0026
```

（3）根据神经网络工具箱函数 trainlm( ) 编写的程序如下：

```
X=-1:0.1:1;T=sin(pi*X);                %提供训练集和目标值
plot(X,T,'+');
%建立网络，并得权值和偏值
S1=5;[W1,b1,W2,b2]=initff(X,S1,'tansig',T,'purelin');
%利用不含噪声的理想输入数据训练网络，并得权值和偏值
disp_freq=10;                          %显示间隔
max_epoch=8000;                        %训练时间
err_goal=0.02;                         %训练目标误差
tp=[disp_freq  max_epoch  err_goal];
[W1,b1,W2,b2,te,tr]=trainlm(W1,b1,'tansig',W2,b2,'purelin',X,T,tp);
X1=0.5;
[y1,y2]=simuff(X1,W1,b1,'tansig',W2,b2,'purelin');
```

```
y2
```

其结果显示为

```
y2 =
    0.9987
```

12. 建立网络函数 newff( )

利用 newff( )函数可建立一个感知机神经网络。其调用格式为

net=newff(Xr,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)

式中，Xr 为一个输入向量，它决定了输入向量的最小值和最大值的取值范围；[S1 S2...SNl] 表示网络隐含层和输出层神经元的个数；{TF1 TF2...TFNl}表示网络隐含层和输出层的传输函数，默认为'tansig'；BTF 表示网络的反向训练函数，默认为'trainlm'；BLF 表示网络的反向权值学习函数，默认为'learngdm'；PF 表示性能数，默认为'mse'；net 为生成的新 BP 神经网络。例如，建立一个非线性函数的 BP 网络逼近正弦函数可利用以下命令

```
>>X=[-1:0.05:1];T=sin(pi*X); %给定训练集和目标值
>>net=newff(minmax(X),[10 1],{'tansig','purelin'},'trainlm')
>>w=net.LW{2,1};b=net.b{2};
>>Y1=sim(net,X);plot(X,T,'-',X,Y1,'--')
```

未经训练的网络输出与目标值（正弦函数）的比较如图 2-33 所示。网络的权值和偏值是随机的，所以未经训练的网络输出效果很差，而且每次运行结果也不一样。

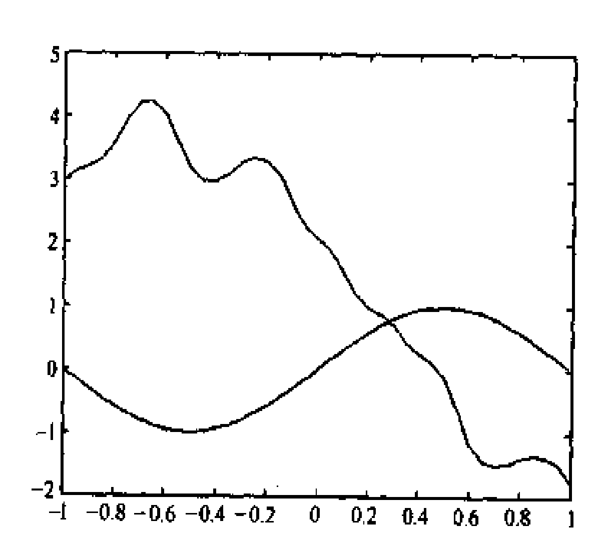


图 2-33 未经训练的网络输出与目标值的比较

例 2-14 利用两层 BP 神经网络训练加权系数。两组 3 输入为[1 2;-1 1;1 3]，希望的输出均为[1,1]。隐含层的激活函数取 S 型传输函数，输出层的激活函数取线性传输函数。

解：根据神经网络工具箱函数编写的程序如下：

```
X=[1 2;-1 1;1 3];T=[1 1;1 1];        %提供两组 3 输入 2 输出的训练集和目标值
```

```
net=newff(minmax(X),[10 2],{'tansig','purelin'},'trainlm');    %建立网络
net=train(net,X,T) ;                      %训练网络
X1=X; Y=sim(net,X1)                   %仿真网络，并给出输出值
```

其结果显示为

```
Y =
     1.0000    1.0000
     1.0000    1.0000
```

13. 计算误差曲面函数 errsurf( )

利用误差曲面函数可以计算单输入神经元误差的平方和。其调用格式为

Es= errsurf (X, T, W,b, 'Tf')

式中，X 为输入向量；T 为目标向量；W 为权值矩阵；b 为偏值向量；Tf 为传输函数。

14. 绘制误差曲面图函数 plotes( )

利用函数 plotes ( ) 可绘制误差曲面图。其调用格式为

plotes(W,b,Es,v)

式中，W 为权值矩阵；b 为偏值向量；Es 为误差曲面；v 为期望的视角，默认值为[−37.5 30]。

例如，利用以下命令可得到如图 2-34 所示的误差曲面图。

```
>>X=[3 2];T=[0.4 0.8];
>>W=-4:0.4:4;b=W;
>>Es=errsurf (X, T, W,b,'logsig');
>>plotes(W,b,Es,[60 30])
```

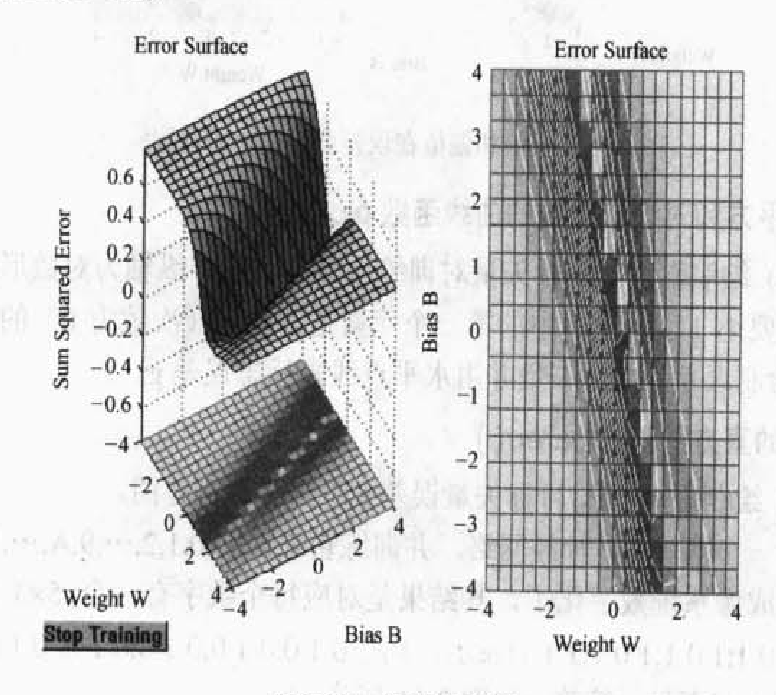


图 2-34　误差曲面图

15. 在误差曲面图上绘制权值和偏值的位置函数 plotep ( )

函数 plotep ( )在已由函数 plotes ( )产生的误差性能表面图上画出单输入网络权值 W 与偏差 b 所对应的误差 e 的位置。该函数的调用格式为：

h=plotep (W,b,e)

式中，W 为权值矩阵；b 为偏值向量；e 为神经元误差。例如，利用以下命令可得到如图 2-35 所示的权值和偏值在误差曲面图上的位置。

```
>>X=[3 2];T=[0.4 0.8]; W=-4:0.4:4;b=W;
>>Es=errsurf (X,T,W,b,'logsig'); plotes(W,b,Es,[60 30])
>>W=-2;b=0;
>>e=sumsqr(T-simuff(X,W,b,'logsig'));plotep(W,b,e)
```

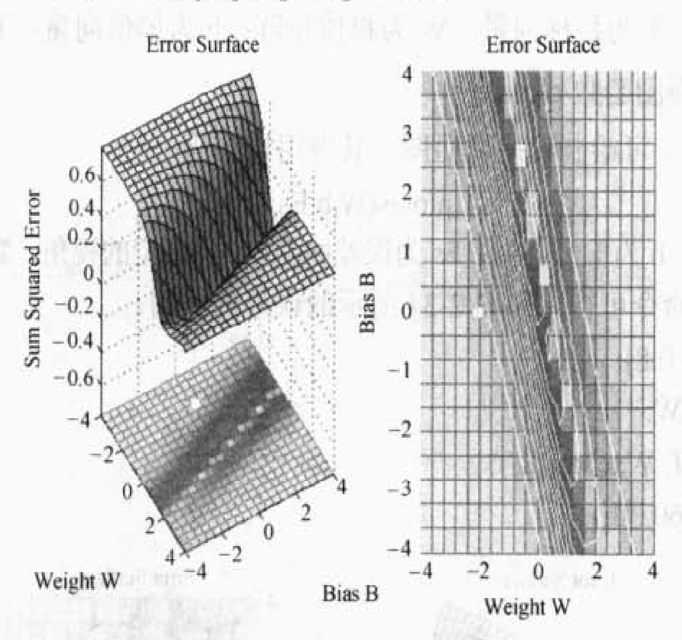


图 2-35 权值和偏值在误差曲面图上的位置

16. 绘制误差平方和对训练次数的曲线函数 ploterr ( )

函数 ploterr (e) 绘制误差 e 的行矢量对训练次数的曲线，纵轴为对数形式。总的训练次数比误差 e 的长度要小 1。误差 e 中的第一个元素是训练前（次数为 0）的初始网络误差。函数 ploterr (e,g) 绘制误差 e 的行矢量并用水平点线来标志误差 g。

17. 绘制误差的直方图函数 barerr( )

函数 barerr (e) 绘制每对输入/目标矢量误差 e 平方和的直方图。

例 2-15 设计一个两层 BP 神经网络，并训练它来识别 0,1,2,…,9,A,…,F。这 16 个十六进制数已经被数字成像系统数字化了，其结果是对应每个数字有一个 5×3 的布尔量网络。例如，0 用[1 1 1;1 0 1;1 0 1;1 0 1;1 1 1]表示；1 用[0 1 0;0 1 0;0 1 0;0 1 0; 0 1 0]表示；2 用[1 1 1; 0 0 1;0 1 0;1 0 0;1 1 1]表示；等等，如图 2-36 所示。

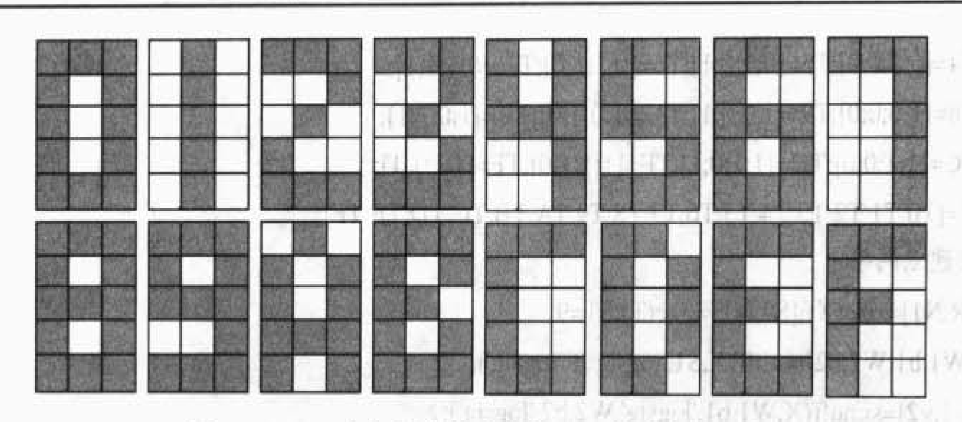


图 2-36　16 个十六进制对应的 5×3 的布尔量网络

解：将这 16 个含 15 个布尔量网络元素的输入向量定义成一个 15×16 维的输入矩阵 X，X 中每一列的 15 个元素对应一个数字量按列展开的布尔量网络元素。例如，X 中第一列的 15 个元素[1;1;1;1;0;1;1;0;1;1;0;1;1;1;1]表示 0。目标向量也被定义成一个 4×16 维的目标矩阵 T，其每一列的 4 个元素对应一个数字量，这 16 个数字量用其所对应的十六进制值表示。例如，用[0;0;0;0]表示 0；用[0;0;0;1]表示 1；用[0;0;1;0]表示 2；等等。

为了识别这些以 5×3 布尔量网络表示的十六进制数，所设计的网络需要有 16 个输入，在输出层需要有 4 个神经元来识别它，隐含层（对应于 MATLAB 工具箱中的第一层）设计了 9 个神经元。激活函数选择 Log-Sigmoid 型传输函数，因为它的输出范围（0～1）正好适合在学习后输出布尔值。由于十六进制数的表示有时会受到噪声的污染，使布尔量网络元素发生变化，故为了能排除噪声的干扰，顺利地识别 16 个十六进制数，必须设计高性能的神经网络。

方法一：根据 initff( )函数编写的 MATLAB 程序如下：

```
%定义用 5×3 布尔量网络表示的 16 个十六进制数 0,1,2,…,9,A,…,F
X0=[1 1 1;1 0 1;1 0 1;1 0 1;1 1 1];X1=[0 1 0;0 1 0;0 1 0;0 1 0; 0 1 0];
X2=[1 1 1;0 0 1;0 1 0;1 0 0;1 1 1]; X3=[1 1 1;0 0 1;0 1 0;0 0 1;1 1 1];
X4=[1 0 1;1 0 1;1 1 1;0 0 1;0 0 1]; X5=[1 1 1;1 0 0;1 1 1;0 0 1;1 1 1];
X6=[1 1 1;1 0 0;1 1 1;1 0 1;1 1 1]; X7=[1 1 1;0 0 1;0 0 1;0 0 1;0 0 1];
X8=[1 1 1;1 0 1;1 1 1;1 0 1;1 1 1]; X9=[1 1 1;1 0 1;1 1 1;0 0 1;1 1 1];
XA=[0 1 0;1 0 1;1 0 1;1 1 1;1 0 1]; XB=[1 1 1;1 0 1;1 1 0;1 0 1;1 1 1];
XC=[1 1 1;1 0 0;1 0 0;1 0 0;1 1 1]; XD=[1 1 1;1 0 1;1 0 1;1 0 1;1 1 0];
XE=[1 1 1;1 0 0;1 1 0;1 0 0;1 1 1]; XF=[1 1 1;1 0 0;1 1 0;1 0 0;1 0 0];
%将每个用 5×3 布尔量网络表示的数按列表示，生成输入矩阵 X
X=[X0(:) X1(:) X2(:) X3(:) X4(:) X5(:) X6(:) X7(:) X8(:) X9(:) …
   XA(:) XB(:) XC(:) XD(:) XE(:) XF(:)];
%定义每个十六进制数对应的目标向量，生成目标矩阵 T
T0=[0;0;0;0];T1=[0;0;0;1];T2=[0;0;1;0];T3=[0;0;1;1];
```

```
T4=[0;1;0;0];T5=[0;1;0;1];T6=[0;1;1;0];T7=[0;1;1;1];
T8=[1;0;0;0];T9=[1;0;0;1];TA=[1;0;1;0];TB=[1;0;1;1];
TC=[1;1;0;0];TD=[1;1;0;1];TE=[1;1;1;0];TF=[1;1;1;1];
T=[T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 TA TB TC TD TE TF];
%建立网络
[R,N1]=size(X);[S2,N1]=size(T);S1=9;
[W1,b1,W2,b2]=initff(X,S1,'logsig',T,'logsig');
[y1,y2]=simuff(X,W1,b1,'logsig',W2,b2,'logsig');
%利用不含噪声的理想输入数据训练网络
disp_freq=20;                    %显示间隔
max_epoch=5000;                  %训练时间
err_goal=0.0001;                 %训练目标误差
mc=0.95;                         %动量参数
tp=[disp_freq max_epoch err_goal NaN NaN NaN mc];
[W1,b1,W2,b2,te,tr]=trainbpx(W1,b1,'logsig',W2,b2,'logsig',X,T,tp);
[y1,y2]=simuff(X,W1,b1,'logsig',W2,b2,'logsig');
%为使网络对输入有一定的容错能力，再利用不含和含有噪声的输入数据训练网络
max_epoch=500;                   %训练时间
err_goal=0.6;                    %训练目标误差
T1=[T T T T];
tp=[disp_freq   max_epoch   err_goal]
for i=1:10
X1=[X X (X+randn(R,N1)*0.1) (X+randn(R,N1)*0.2)];
[W1,b1,W2,b2,te,tr]=trainbpx(W1,b1,'logsig',W2,b2,'logsig',X1,T1,tp);
end
[y1,y2]=simuff(X1,W1,b1,'logsig',W2,b2,'logsig');
%为了保证网络总能够正确对理想输入信号进行识别，再次用理想信号进行训练
disp_freq=20;                    %显示间隔
max_epoch=5000;                  %训练时间
err_goal=0.001;                  %训练目标误差
tp=[disp_freq max_epoch err_goal];
[W1,b1,W2,b2,te,tr]=trainbpx(W1,b1,'logsig',W2,b2,'logsig',X,T,tp);
X1=X(:,2:2:16);                  %定义网络的输入为十六进制数 1,3,5,7,9,B,D,F
y=simuff(X1,W1,b1,'logsig',W2,b2,'logsig')
```

其结果显示为

```
y =
```

```
0.0000  0.0000  0.0000  0.0000  0.9999  1.0000  0.9999  1.0000
0.0002  0.0000  1.0000  1.0000  0.0001  0.0236  1.0000  1.0000
0.0008  1.0000  0.0000  1.0000  0.0000  1.0000  0.0000  0.9997
1.0000  0.9995  0.9985  1.0000  1.0000  0.9964  1.0000  1.0000
```

方法二：根据 newff( )函数编写的 MATLAB 程序如下：

```
%定义用 5×3 布尔量网络表示的 16 个十六进制数 0,1,2,…,9,A,…,F
X0=[1 1 1;1 0 1;1 0 1;1 0 1;1 1 1];X1=[0 1 0;0 1 0;0 1 0;0 1 0; 0 1 0];
X2=[1 1 1;0 0 1;0 1 0;1 0 0;1 1 1]; X3=[1 1 1;0 0 1;0 1 0;0 0 1;1 1 1];
X4=[1 0 1;1 0 1;1 1 1;0 0 1;0 0 1]; X5=[1 1 1;1 0 0;1 1 1;0 0 1;1 1 1];
X6=[1 1 1;1 0 0;1 1 1;1 0 1;1 1 1]; X7=[1 1 1;0 0 1;0 0 1;0 0 1;0 0 1];
X8=[1 1 1;1 0 1;1 1 1;1 0 1;1 1 1]; X9=[1 1 1;1 0 1;1 1 1;0 0 1;1 1 1];
XA=[0 1 0;1 0 1;1 0 1;1 1 1;1 0 1]; XB=[1 1 1;1 0 1;1 1 0;1 0 1;1 1 1];
XC=[1 1 1;1 0 0;1 0 0;1 0 0;1 1 1]; XD=[1 1 1;1 0 1;1 0 1;1 0 1;1 1 0];
XE=[1 1 1;1 0 0;1 1 0;1 0 0;1 1 1]; XF=[1 1 1;1 0 0;1 1 0;1 0 0;1 0 0];
%将每个用 5×3 布尔量网络表示的数按列表示成一列元素，生成输入矩阵 X
X=[X0(:) X1(:) X2(:) X3(:) X4(:) X5(:) X6(:) X7(:) X8(:) X9(:) ...
    XA(:) XB(:) XC(:) XD(:) XE(:) XF(:)];
%定义每个十六进制数对应的目标向量，生成目标矩阵 T
T0=[0;0;0;0];T1=[0;0;0;1];T2=[0;0;1;0];T3=[0;0;1;1];
T4=[0;1;0;0];T5=[0;1;0;1];T6=[0;1;1;0];T7=[0;1;1;1];
T8=[1;0;0;0];T9=[1;0;0;1];TA=[1;0;1;0];TB=[1;0;1;1];
TC=[1;1;0;0];TD=[1;1;0;1];TE=[1;1;1;0];TF=[1;1;1;1];
T=[T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 TA TB TC TD TE TF];
%建立网络，并得权值和偏值
[R,N1]=size(X);[S2,N1]=size(T);S1=9;
net=newff(minmax(X),[S1 S2],{'logsig','logsig'},'traingdx');
w=net.LW{2,1};
b1=net.b{1};b2=net.b{2};
y1=sim(net,X);
%利用不含噪声的理想输入数据训练网络，并得权值和偏值
net.performFcn='sse';                 %平方和误差函数
net.trainParam.goal=0.000001;         %训练目标误差
net.trainParam.epochs=5000;           %训练时间
net.trainParam.show=20;               %计算步长
net.trainParam.mc=0.95;               %冲量参数
```

```
[net,tr]=train(net,X,T);
w=net.LW{2,1};
b1=net.b{1};b2=net.b{2};
y2=sim(net,X);
%为使网络对输入有一定的容错能力，再利用不含和含有噪声的输入数据训练网络
net.trainParam.goal=0.6;            %训练目标误差
net.trainParam.epochs=500;          %训练时间
net1=net;
T1=[T T T T];
for i=1:10
X1=[X X (X+randn(R,N1)*0.1) (X+randn(R,N1)*0.2)];
[net1,tr]=train(net1,X1,T1)
end
y3=sim(net1,X);
%为了保证网络总能够正确对理想输入信号进行识别，再次用理想信号进行训练
[net1,tr]=train(net1,X,T);
w=net.LW{2,1};
b1=net.b{1};b2=net.b{2};
X1=X(:,2:2:16);
y4=sim(net1,X1)
```

其结果显示为

```
y4 =
    0.0000  0.0000  0.0004  0.0000  1.0000  1.0000  1.0000  1.0000
    0.0001  0.0011  0.9992  0.9994  0.0004  0.0000  1.0000  0.9999
    0.0010  0.9993  0.0040  1.0000  0.0000  0.9999  0.0000  1.0000
    1.0000  1.0000  1.0000  1.0000  1.0000  0.9997  1.0000  1.0000
```

由以上结果可知，网络的输出为十六进制数的 1,3,5,7,9,B,D,F，刚好为网络给定的输入，表明网络训练成功。

2.4 径向基神经网络工具箱函数

MATLAB 神经网络工具箱提供了大量的与径向基网络相关的工具箱函数。在 MATLAB 工作空间的命令行输入“help radbasis”，便可得到与径向基神经网络相关的函数，进一步利用 help 命令又能得到相关函数的详细介绍。表 2-4 列出了径向基网络的重要函数和基本功能。

表 2-4　径向基网络的重要函数和基本功能

函数名	功能
dist( )	计算矢量间的距离函数
radbas( )	径向基传输函数
solverb( )	设计一个径向基神经网络
solverbe( )	设计一个精确径向基神经网络
simurb( )	径向基神经网络仿真函数
newrb( )	新建一个径向基神经网络
newrbe( )	新建一个严格的径向基神经网络
newgrnn( )	新建一个广义回归径向基神经网络
newpnn( )	新建一个概率径向基神经网络
mse( )	均方误差性能函数
ind2vec( )	将下标矢量变换成单值矢量组
vec2ind ( )	将单值矢量组变换成下标矢量

1. 计算矢量间的距离函数 dist( )

大多数神经元网络的输入可通过表达式 N=w*X+b 来计算。其中，w，b 分别为权矢量和偏差矢量。但有一些神经元的输入可由函数 dist( )来计算。dist( )函数是一个欧氏(Euclidean) 距离权值函数。它对输入进行加权，得到被加权的输入。一般两个向量 x 和 y 之间的欧氏(Euclidean)距离 d 被定义为：d=sun((x−y).^2).^0.5。函数 dist( )的调用格式为

d=dist(W,X)

或

d=dist(pos)

式中，W 为 $S\times R$ 权值矩阵；X 为 $R\times Q$ 输入矩阵；d 为 $S\times Q$ 的输出距离矩阵，其中含有 W 的权值（行）矢量和 X 的输入（列）矢量的矢量距离。d=disk(pos) 函数也可以作为一个阶层距离函数，用于查找某一层神经网络中的所有神经元之间的欧氏距离。函数也返回一个距离矩阵，如

```
>>w=rand(4,3);X=rand(3,1);d=dist(w,X)
```

其结果显示为

```
d =
     0.7269
     0.7035
     0.8741
     0.7699
```

2. 径向基传输函数 radbas( )

径向基函数神经元的传输函数为 radbas( )。RBF 网络的输入同前面介绍的神经网络的表

达式有所不同。其网络输入为权值向量 W 与输入向量 X 之间的向量距离乘以偏值 b，即 d=radbas(dist(W,X)*b)。该函数的调用格式为

a=radbas(N)

或

a=radbas(Z,b)

a=radbas (P)

函数 radbas (N) 将径向基传输函数作用于网络输入矩阵 N 的每一个元素，返回网络输入向量 N 的输出矩阵 a；函数 radbas (Z,b)用于矢量是成批处理且存在偏差的情况下，此时的偏差 b 和加权输入矩阵 Z 是分开传输的，偏差矢量 b 与 Z 中的每个矢量进行元素相乘，形成网络输入矩阵，函数 radbas( ) 作用于网络输入矩阵的每个元素；函数 radbas (P)包含传输函数的特性名并返回问题中的特性。如下的特性可从任何传输函数中获得：

（1）delta——与传输函数相关的 delta 函数；

（2）init——传输函数的标准初始化函数；

（3）name——传输函数的全称；

（4）output——包含有传输函数最小、最大值的二元矢量。

例如，利用以下命令可得到如图 2-37 所示的径向基传输函数。

```
>>n=-5:0.1:5;a=radbas(n);plot(n,a)
```

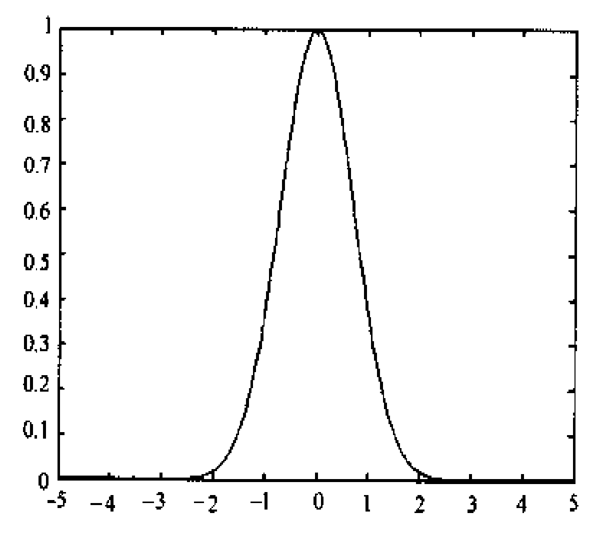


图 2-37 径向基传输函数

3. 设计一个径向基网络函数 solverb( )

径向基网络是一个由径向基神经元隐含层和一个线性神经元输出层组成的两层神经网络。径向基网络不仅能较好地拟合任意不连续的函数，而且能用快速的设计来代替训练。利用函数 solverb( )设计的径向基神经网络，因在建立网络时预先设置了目标参数，故它同时也完成了网络的训练，即可以不经过训练，直接使用。该函数的调用格式为：

[W1,b1,W2,b2,nr,dr]=solverb(X,T,dp)

式中，X 为输入向量；T 为目标向量；dp 是设计参数，可以默认，其中 tp(1)显示间隔次数，默认值为 25；dp(2)为最大的神经元数，默认值为 1000；dp(3)为目标误差，默认值为 0.02；dp(4)为径向基层的散布，默认值为 1.0；W1 和 b1 为网络的径向基神经元隐含层的权值和偏值；W2 和 b2 为网络的输出层的权值和偏值；nr 为径向基函数网络中 radbas 层的神经元个数；dr 为设计误差。

4. 设计一个精确径向基网络函数 solverbe( )

函数 solverbe( )产生一个与输入向量 X 一样多的隐含层径向基神经网络，因而网络对设计的输入/目标向量集误差为 0。该函数的调用格式为

[W1,b1,W2,b2]=solverbe(X,T,sc)

式中，X 为输入向量；T 为目标向量；sc 是径向基函数的宽度，即从函数顶点 1～0.5 的距离，默认值为 1；W1 和 b1 为网络的径向基神经元隐含层的权值和偏值；W2 和 b2 为网络的输出层的权值和偏值。

5. 径向基网络仿真函数 simurb( )

径向基网络设计和训练好以后便可对网络进行仿真。其调用格式为

y=simurb (X,W1,b1,W2,b2)

式中，X 为网络的输入向量；W1 和 b1 为网络的径向基神经元隐含层的权值和偏值；W2 和 b2 为网络的输出层的权值和偏值；y 为网络输出。

例 2-16　利用径向基网络完成函数逼近。

解：根据神经网络函数编写的程序如下：

```
X=-1:0.1:1;T=sin(pi*X); %提供训练集和目标值
plot(X,T,'+');
%建立网络
disp_freg=10;          %显示间隔
max_neuron=100;     %最多的神经元数
err_goal=0.02;        %目标误差
sc=1;                   %径向基函数的分布常数
dp=[disp_freg max_neuron err_goal sc];
[W1,b1,W2,b2]=solverb(X,T,dp);
X1=0.5;y=simurb(X1,W1,b1,W2,b2)
```

利用以上程序可得到如图 2-38 所示的训练后的输出与目标值曲线及如下结果。

```
y =
     0.9977
```

本例采用径向基函数网络来完成函数逼近任务，将结果同 BP 网络及改进 BP 算法的前向网络的训练结果进行比较后，发现径向基函数网络所用的时间最短。

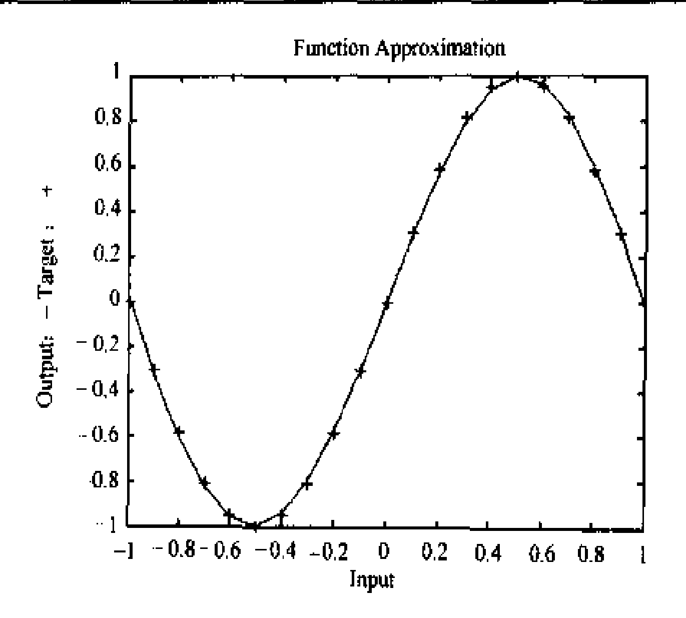


图 2-38 训练后的输出与目标值

6. 新建一个径向基网络函数 newrb( )

该函数的调用格式为

net=newrb(X,T,goal,spread)

式中，X 为输入向量；T 为目标向量；goal 为均方误差，默认为 0；spread 为径向基函数的分布，默认值为 1；net 为生成的网络。利用函数 newrb( )新建的径向基神经网络，也可以不经过训练直接使用。例如，建立一个径向基网络可利用以下命令，即

```
>>X=[1 2 3];T=[2.0 4.1 5.9];
>>net=newrb(X,T);y=sim(net,X)
```

其结果显示为：

```
y =
      2.0000    4.1000    5.9000
```

函数 newrbe( )、newgrnn( )和 newpnn( )的用法同函数 newrb( )。

7. 均方误差性能函数 mse( )

均方误差性能函数的调用格式为

perf=mse(e,p,pp)

式中，e 为误差矩阵或向量；p 为所有权值和偏值向量（可忽略）；pp 为性能参数（可忽略）。

8. 将下标矢量变换成单值矢量组函数 ind2vec( )

该函数的调用格式为

vec=ind2vec(ind)

式中，ind 为包含 n 个下标的行矢量 x；vec 为 m 行 n 列的矢量组矩阵，矩阵中的每个矢量 i，除了由 x 中的第 i 个元素指定的位置为 1 外，其余元素均为 0，矩阵的行数 m 等于 x 中最大的下标值，如

```
>>ind=[1 3 2 3];vec=ind2vec(ind)
```

其结果显示为：

```
vec =
      (1,1)        1
      (3,2)        1
      (2,3)        1
      (3,4)        1
```

9. 将单值矢量组变换成下标矢量函数 vec2ind ( )

函数 vec2ind ( )与函数 ind2vec( )互为逆变化。函数 vec2ind ( )的调用格式为

ind=vec2ind (vec)

式中，vec 为 m 行 n 列的矢量组矩阵 x，x 中的每个矢量 i，除包含一个 1 外，其余元素均为 0，得到的行矢量包括这些非零元素的下标；ind 为 n 个下标值大小等于 0 的行矢量。

例 2-17　利用径向基网络实现函数逼近。

解：MATLAB 程序如下：

```
%给定要逼近的函数样本
X=-1:0.1:1;T=sin(X*pi);
%径向基传输函数及其加权和
n=-3:0.1:3;a1=radbas(n);a2=radbas(n-1.5);a3=radbas(n+2);
a=a1+1*a2+0.5*a3;figure;plot(n,a1,n,a2,n,a3,n,a,'x');
net=newrb(X,T,0.02,1);                 %建立网络
X1=-1:0.01:1;y=sim(net,X1);            %仿真网络
figure;plot(X1,y,X,T,'+');             %绘制网络预测输出及其误差
```

其执行结果如图 2-39 和图 2-40 所示。

利用函数 newrb( )建立的径向基网络能够在给定的误差目标范围内找到能解决问题的最小的网络。但并不等于径向基网络就可以取代其他前馈网络。这是因为径向基网络很可能需要比 BP 网络多得多的隐含层神经网络元来完成工作。BP 网络使用 sigmoid( )函数，这样的神经元有很大的输入可见区域。而径向基网络使用的径向基函数，输入空间区域很小。这就导致了在实际需要的输入空间较大时，需要很多的径向基神经元。

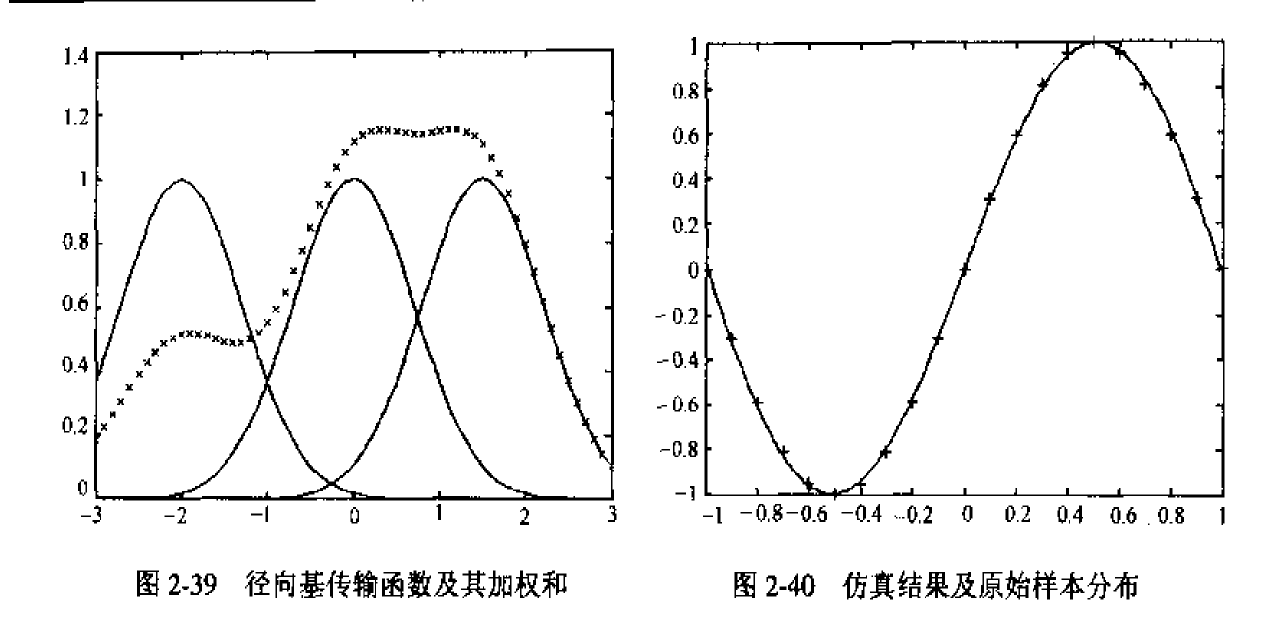


图 2-39 径向基传输函数及其加权和

图 2-40 仿真结果及原始样本分布

2.5 自组织神经网络工具箱函数

自组织神经网络是神经网络领域中最吸引人的话题之一。这种结构的网络能够从输入信息中找出规律及关系，并且根据这些规律来相应地调整均衡网络，使得以后的输出与之相适应。自组织竞争神经网络能够识别成组的相似向量，常用于进行模式分类。自组织特征映射神经网络不但能够像自组织竞争神经网络一样学习输入的分布情况，而且可以学习进行训练神经网络的拓扑结构。在 MATLAB 神经网络工具箱中，自组织网络被分为自组织竞争网络和自组织特征映射神经网络两种。在 MATLAB 工作空间的命令行输入"help selforg"，便可得到与自组织网络相关的函数，进一步利用 help 命令又能得到相关函数的详细介绍。表 2-5 列出了自组织神经网络的重要函数的名称和基本功能。

表 2-5 自组织神经网络的重要函数和基本功能

函 数 名	功 能
compet( )	竞争传输函数
nngenc( )	产生一定类别的样本向量
nbdist( )	用矢量距离表示的领域矩阵
nbgrid( )	用栅格距离表示的领域矩阵
nbman( )	用 Manhattan 距离表示的领域矩阵
plotsm( )	绘制竞争网络的权值矢量
initc( )	初始化竞争神经网络
trainc( )	训练竞争神经网络
simuc( )	仿真竞争神经网络
newc( )	建立一个竞争神经网络

续表

函 数 名	功 能
initsm( )	初始化自组织特征映射网络
learnk ( )	Kohonen 权值学习规则函数
learnis ( )	Instar 权值学习规则函数
learnos ( )	Outstar 权值学习规则函数
learnh ( )	Hebb 权值学习规则函数
learnhd( )	衰减的 Hebb 权值学习规则函数
learnsom( )	自组织特征映射权值学习规则函数
plotsom( )	绘制自组织特征映射网络的权值矢量
trainsm( )	利用 Kohonen 规则训练自组织特征映射网络
simusm( )	仿真自组织特征映射网络
newsom( )	创建一个自组织特征映射神经网络
dist( )	欧氏距离权值函数
mandist( )	Manhattan 距离权值函数
linkdist( )	Link 距离权值函数
midpoit ( )	中点权值初始化函数
negdist( )	对输入矢量进行加权计算
netsum( )	计算网络输入矢量和

1. 竞争传输函数 compet( )

函数 compet( )将神经网络输入进行转换，使网络输入最大的神经元输出为 1，而其余的神经元输出为 0。函数 compet ( )的调用格式为

Y=compet(X)

或

Y=compet(Z,b)

其中，compet(X)返回的输出矢量矩阵 Y，其每一列中仅包含一个 1，位于对应的响应矢量在网络输入向量矩阵 X 中有最大值的位置，而其余的元素均为 0。compet(Z,b)用于成批处理矢量且存在偏差的情况下，偏差矢量 b 附加到加权输入矩阵 Z 的每一个矢量上，形成网络输入矢量矩阵 X，然后利用竞争传输函数将输入矢量转换为输出矢量矩阵 Y，如

```
>>X=[0;0.2;0.6;0.1];Y=compet(X)
```

其结果显示为：

```
Y =

     (3,1)        1
```

很明显，输出向量 y 在网络输入向量 X 中的最大元素 0.6 处输出为 1。此处，y 是以稀疏矩阵的形式返回的，这种形式很有效，因为只需存储元素为 1 的位置。用函数 full( ) 可以看它的非稀疏形式。例如，利用以下命令可查看 y 的全部内容，即

```
>>X=[0;0.2;0.6;0.1];Y=compet(X);full(Y)
```

其结果显示为

```
ans =
        0
        0
        1
        0
```

2. 产生一定类别的样本向量函数 nngenc( )

函数 nngenc( )的调用格式为

X=nngenc(C,clusters,points,std_dev)

其中，C 指定类中心范围；clusters 指定类别数目；points 指定每一类的样本点的数目；std_dev 指定每一类的样本点的标准差；X 为产生的样本向量。

例如，指定类中心范围为 0～1，类别数目为 5，每一类别有 10 个样本点，每一类的样本点的标准差为 0.05，则可利用以下命令得图 2-41。

```
>>C=[0 1;0 1];
>>clusters=5;points=10;std_dev=0.05;
>>X=nngenc(C,clusters,points,std_dev);
>>plot(X(1,:),X(2,:),'+r')
```

图 2-41 显示了这些输入样本点的分布情况。这些随机产生的样本向量分成了五类。

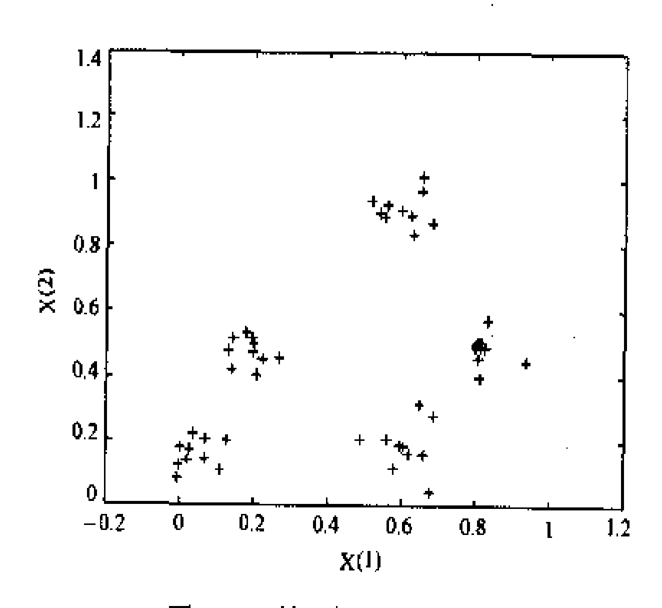


图 2-41　输入样本向量的分布

3. 用矢量距离表示的邻域矩阵函数 nbdist( )

自组织网络中的神经元可以按照任何方式排列。这种排列可以用表示同一层神经元间

距离的邻域来描述。该函数的调用格式为

m=nbdist(d)

或　　m=nbdist(d1,d2,…,d5)

式中，d,d1,d2,…,d5 为神经元的个数；m 为网络的邻域。nbdist(d)返回一维排列的 d×d 的邻域，表示一维中含有 *d* 个神经元，其中第 *i* 行第 *j* 列的元素(i,j)表示神经元 i 与神经元 j 之间的距离。nbdist(d1,d2) 返回二维排列的(d1*d2)×(d1*d2)的邻域，表示二维中含有 d1* d2 个神经元，其中第 *i* 行第 *j* 列的元素(i,j)表示神经元 i 与神经元 j 之间的距离。nbdist ( )函数可用于每层最多有 5 维、最多有 5 层的网络，一般而言维数越少，收敛越快，如

```
>>d=nbdist(2,3)
```

其结果显示为

```
m =
     0     1     2     1     1     2
     1     0     1     1     1     1
     2     1     0     2     1     1
     1     1     2     0     1     2
     1     1     1     1     0     1
     2     1     1     2     1     0
```

4. 用栅格距离表示的邻域矩阵函数 nbgrid( )

两神经元的栅格距离是指在神经元坐标相减后的矢量中元素幅值的最大值。该函数的调用格式为

m=nbgrid (d1)

或　　m=nbgrid (d1,d2,…,d5)

nbgrid( )函数可用于每层最多有 5 维、最多有 5 层的网络。一般而言，维数越少，收敛越快。用法同函数 nbdist( )。例如，利用函数 nbgrid( )可产生一个 2×3（6 个神经元）的二维排列的邻域，即

```
>> m=nbgrid (2,3)
```

5. 用 Manhattan 距离表示的邻域矩阵函数 nbman( )

两神经元的 Manhattan 距离是指在神经元坐标相减后的矢量中的元素的绝对值之和。该函数的调用格式为

m=nbman (d1)

或　　m=nbman (d1,d2,…,d5)

6. 绘制竞争网络的权值矢量函数 plotsm( )

函数 plotsm(W,M)用于绘制自组织竞争网络的权值图，在每个神经元的权矢量（行）相

应的坐标处画一点，表示相邻神经元权值的点，根据邻阵 M 用实线连接起来，即如果 M(i,j) 小于等于 1，则将神经元 i 和 j 用线连接起来。其调用格式为

plotsm(W,M,nd)

例如，对两组输入为 12 个神经元，随机产生权值，可利用以下命令得到如图 2-42 所示的输入样本向量的分布。

```
>>W=rands(12,2); M=nbman(3,4);plotsm(W, M)
```

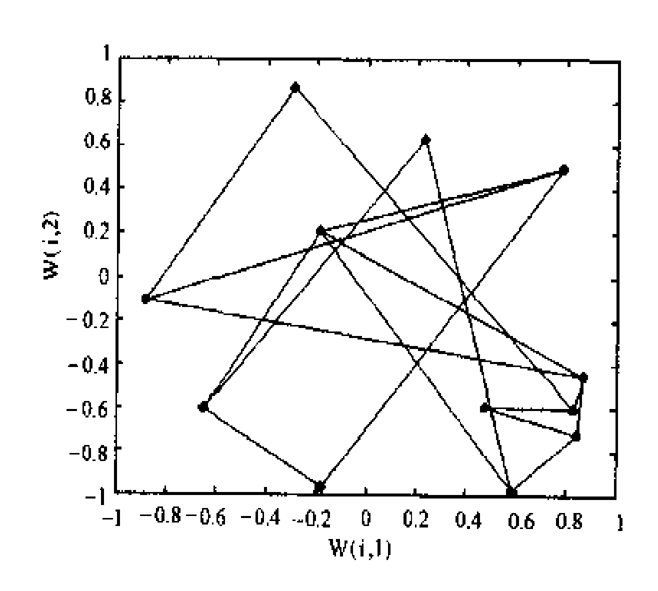


图 2-42　输入样本向量的分布

7. **初始化竞争层函数 initc( )**

该函数的调用格式为

w=initc(X,S)

式中，X 为输入矢量矩阵，X 的第 i 行中应包含网络输入 i 的所有可能的最小值和最大值，这样的权值 w 才能正确的初始化；S 为竞争层神经元数；w 为竞争层的权值。例如，网络有两个输入，变化范围分别为[-3,3]和[0,6]，竞争层有 5 个神经元，利用函数 initc( )初始化竞争层权值，可利用以下命令，即

```
>>X=[-3 3;0 6];w=initc(X,5)
```

其结果显示为

```
w =
      0     3
      0     3
      0     3
      0     3
      0     3
```

8. 训练竞争层函数 trainc( )

训练竞争层函数 trainc( )对一组输入矢量分类，它从一组输入矢量中随机选取一个矢量，然后找出输入最大的神经元，并按 Kohonen 准则修改权值，从而训练竞争层网络。该函数的调用格式为

[W,b]=trainc(w,X,tp)

式中，w 和 W 分别为训练前后的权值矩阵；b 为训练后的偏值向量；X 为输入向量；tp 是训练控制参数，其中 tp(1)为更新显示的样本数，默认值为 25；tp(2)为训练样本的总次数，默认值为 100；tp(3)为学习速率，默认值为 0.01；tp(4)为跟踪性能（从 0～1），默认值为 0.999；tp(5)为加权性能（从 0～1），默认值为 0.1。

例如，对于一个三维标准化的二元输入矢量组，现以每 10 步显示一次，最大训练步数为 500 步，学习速率为 0.1，训练一个竞争层有三个神经元的竞争网络可利用以下命令，即

```
>>X=[0.5827 0.6496 -0.7798;0.8127 0.7603 0.6260];
>>w=initc(X,3);
>>[w,b]=trainc(w,X,[10 500 0.1])
```

其结果显示为

```
w =
      -0.7798     0.6260
       0.5846     0.8112
       0.6310     0.8749
```

输出结果表示，神经元 1 的权矢量（权矩阵的第一行）学会了第三个输入矢量。神经元 2 的权矢量在第一个和第二个输入矢量之间，这些矢量靠得非常近，所以网络“决定”把它们分为一类，即把它们分给同一个神经元。神经元 3 在这个短期训练过程中，没有靠近任何一个输入矢量而赢得权竞争，因而它什么也未学会，变成所谓的“死”神经元。与第一个或第二个输入矢量相似的新矢量的出现将导致神经元 2 输出一个 1，与第三个输入矢量相似的新矢量将导致神经元 1 输出 1。

9. 竞争网络仿真函数 simuc( )

一个竞争层包含一层神经元，在任何给定的时间，只有网络输入最大的神经元输出为 1，其他的神经元为 0。由于竞争层的任何一个输出向量中仅含惟一的非零值，因此该函数返回值是一个稀疏矩阵。该函数的调用格式为

Y=simuc(X,w)

式中，w 为竞争层权值矩阵；X 为输入矢量；Y 为网络输出矩阵。例如，计算竞争层对输入的响应可利用以下命令，即

```
>>X=[0 2;-5 5];w=initc(X,3);
```

```
>>[w,b]=trainc(w,X,[10 100 0.5]);Y=simuc([2;4],w)
```

其结果显示为

```
Y=
    (1,1)   1
```

输出结果表示神经元 1 有一个输出 1。

例 2-18 建立一个输入向量分布在一个二维空间，其变化范围分别为[0 1]和[0 1]，用来区分 5 种模式的自组织竞争神经网络，并对其进行训练和仿真。

解：MATLAB 程序如下：

```
%产生具有 5 类样本类别的样本点，并在图中绘制出
C=[0 1;0 1];clusters=5;points=6;std_dev=0.05;
X=nngenc(C,clusters,points,std_dev);
plot(X(1,:),X(2,:),'+r');xlabel('X(1)') ;ylabel('X(2)')
%建立神经元为 5 的一个自组织竞争神经网络，并求出初始权值
w=initc([0 1;0 1],5);
figure;plot(X(1,:),X(2,:),'+r',w(:,1),w(:,2),'ow');
xlabel('X(1),w(1)') ;ylabel('X(2),w(2)')
%训练神经网络
df=20;                  %显示间隔
me=500;                 %训练步数
lr=0.1;                 %学习速率
tp=[df me lr];
w=trainc(w,X,tp);
%对于训练好的网络进行测试与使用
X1=[0;0.2];y=simuc(X1,w)
```

利用以上程序可得到如图 2-43 所示的训练后网络权值的分布及如下结果，即

```
y =
    (3,1)    1
```

从如图 2-43 所示可见，网络经过训练以后，权值得到了调整，调整后的权值分布在各个类的中心位置上。对于需要分类的模式矢量[0;0.2]，将其输入到训练好的网络中，网络就可以对其分类。分类结果指出了第(3,1)个神经元发生了响应，它反映了这个输入所属的类别。

10. 建立竞争网络函数 newc( )

利用 newc( )函数可建立一个竞争层。其调用格式为

net=newc(Xr,S,lr)

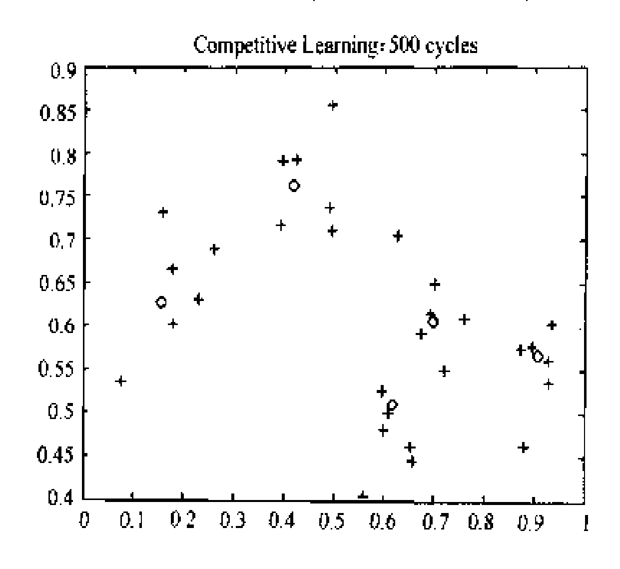


图2-43　训练后网络权值的分布

式中，Xr为一个$R\times2$维的输入矩阵，它决定了输入向量的最小值和最大值的取值范围；R为输入向量的个数；S表示神经元的个数；lr表示Kohonen学习速率，默认值为0.01；net为生成的自组织竞争网络。例如，建立一个输入向量分布在一个二维空间，其变化范围为[0 1]，用来区分5种模式的自组织竞争神经网络，可利用以下命令，即

```
>>net=newc([0 1;0 1],5,0.1);w=net.iw{1}
```

其结果显示为

```
w =
      0.5000    0.5000
      0.5000    0.5000
      0.5000    0.5000
      0.5000    0.5000
      0.5000    0.5000
```

例2-19　建立一个输入向量分布在一个二维空间，其变化范围分别为[0 1]和[0 1]，用来区分5种模式的自组织竞争神经网络，并对其进行训练和仿真。

解：MATLAB程序如下：

```
%产生具有5类样本类别的样本点，并在图中绘制出
C=[0 1;0 1];clusters=5;points=10;std_dev=0.05;
X=nngenc(C,clusters,points,std_dev);
plot(X(1,:),X(2,:),'+r');xlabel('X(1)') ;ylabel('X(2)')
%建立神经元为5的一个自组织竞争神经网络，并求出初始权值
net=newc([0 1;0 1],5,0.1);
w=net.iw{1};                    %初始权值
%训练神经网络，并设置最大训练步数为7
```

```
net.trainParam.epochs=7;      %最大训练步数
net=init(net);net=train(net,X);
w1=net.iw{1};                 %训练后的权值
plot(X(1,:),X(2,:),'+r');
xlabel('X(1)') ;ylabel('X(2)')
hold on;plot(w1(:,1),w1(:,2),'ob');hold off
%对于训练好的网络进行测试与使用
X1=[0.6;0.8];y=sim(net,X1)
```

利用以上程序可得到如图 2-44 所示的训练后网络权值的分布及如下结果，即

```
y =
   (4,1)   1
```

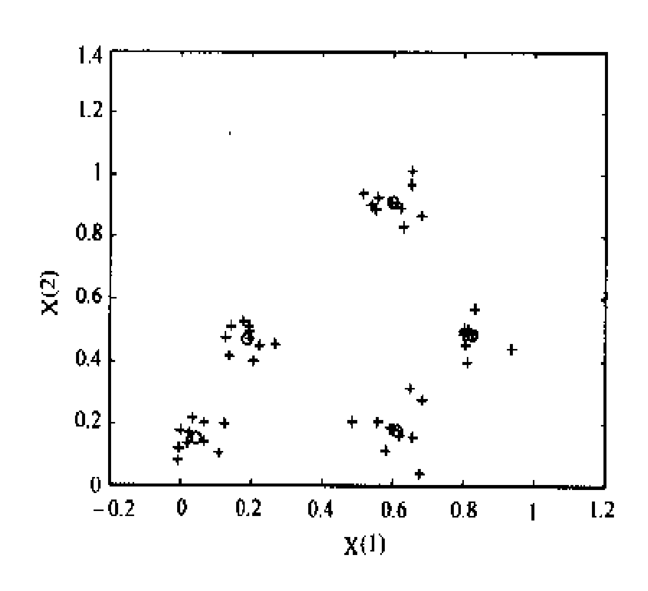


图 2-44 训练后网络权值的分布

从如图 2-44 所示可见，网络经过训练以后，权值得到了调整，调整后的权值分布在各个类的中心位置上。对于需要分类的模式矢量[0.6;0.8]，将其输入到训练好的网络中，网络就可以对其分类。分类结果指出了第(4,1)个神经元发生了响应，它反映了这个输入所属的类别。

11．初始化自组织特征映射网络函数 initsm( )

函数 initsm( )用于对自组织特征映射网络的权值进行初始化，因为自组织特征映射网络不需要偏值。该函数的调用格式为

w=initsm(X,S)

式中，X 为输入矢量矩阵，X 的第 i 行中应包含网络输入 i 的所有可能的最小值和最大值，这样的权值 w 才能正确的初始化；S 为竞争层神经元数；w 为权值。

12. Konohen 权值学习函数 learnk( )

函数 learnk( )根据 Konohen 相关准则计算网络层的权值变化矩阵，其学习通过调整神经元的权值等于当前输入，使神经元存储输入，用于以后的识别，即$\Delta w(i,j)=\eta(x(j)-w(i,j))$。该函数的调用格式为

[dW,NLS]=learnk(W,X,Z,N,A,T,E,gW,gA,D,LP,LS)

式中，W 为 S×R 维的权值矩阵（或 S×1 维的偏值向量）；X 为 Q 组 R 维的输入向量；Z 为 Q 组 S 维的权值输入向量；N 为 Q 组 S 维的网络输入向量；A 为 Q 组 S 维的输出向量；T 为 Q 组 S 维的目标向量；E 为 Q 组 S 维的误差向量；gW 为 S×R 维的性能参数梯度；gA 为 Q 组 S 维的性能参数的输出梯度；D 为权值的修正量；LP 为学习参数，包括学习速率 LP.lr，默认时学习速率为 0.01；LS 为学习状态，初始值为[ ]；dW 为 S×R 维的权值（或偏值）变化矩阵；NLS 为新的学习状态。例如，在给定随机输入矩阵 X，输出矩阵 A，权值矩阵 w 和学习速率 LP 后，可根据以下命令计算其网络层的权值变化矩阵。

```
>>X=rand(3,2);A=rand(3,2);w=rand(3,3);lp.lr=0.5;
>>dw=learnk(w,X,[],[],A,[],[],[],[],[],lp,[])
```

其结果显示为

```
dw =
    0.4463    0.4474    0.4207
   -0.1771    0.5722    0.3211
   -0.0387   -0.3100   -0.0416
```

13. Instar 权值学习函数 learnis( )

函数 learnis( ) 根据 Instar 相关准则计算网络层的权值变化矩阵。其学习用一个正比于神经网络的学习速率来调整权值，学习一个新的矢量使之等于当前输入。这样，任何使 Instar 层引起高输出的变化，都会导致网络根据当前的输入向量学习这种变化，最终相同的输入使网络有明显不同的输出，即$\Delta w(i,j)=\eta\ y(i)\ (x(j)-w(i,j))$。该函数的调用格式为

[dW,NLS]=learnis(W,X,Z,N,A,T,E,gW,gA,D,LP,LS)

该函数的用法和各个参数的定义同函数 learnk( )，如

```
>>w=eye(3);b=-0.5*ones(3,1);X=[1 0;0 1;1 1];A=hardlim(w*X,b);
>>lp.lr=0.5;dw=learnis(w,X,[],[],A,[],[],[],[],[],lp,[])
```

其结果显示为

```
dw =
         0         0    0.5000
         0         0    0.5000
    0.5000    0.5000         0
```

14. Outstar 权值学习函数 learnos( )

函数 learnos( )根据 Outstar 相关准则计算网络层的权值变化矩阵。Outstar 网络层的权值可以看做是与网络层的输入矢量一样多的长期存储器。通常，Outstar 层是线性的，允许输入权值按线性层学习输入矢量。因此，存储在输入权值中的矢量可通过激活该输入而得到，即$\Delta w(i,j)=\eta(y(i)-w(i,j))/x(j)$。该函数的调用格式为：

[dW,NLS]=learnos(W,X,Z,N,A,T,E,gW,gA,D,LP,LS)

该函数的用法和各个参数的定义同函数 learnk( )，如

```
>>X=rand(3,2);A=rand(3,2);w=rand(3,3);lp.lr=0.5;
>>dw=learnos(w,X,[],[],A,[],[],[],[],[],lp,[])
```

其结果显示为

```
dw =
   -0.0509    0.0496   -0.2044
    0.0706    0.2407    0.1263
    0.1858    0.1844    0.3651
```

15. Hebb 权值学习规则函数 learnh( )

Hebb 在 1943 年首次提出了神经元学习规则，认为两个神经元之间的连接权值的强度与所连接的两个神经元的活化水平成正比。也就是说，如果一个神经元的输入值大，那么其输出值也大，而且输入和神经元之间的权值也相应增大。其原理可表示为：$\Delta w(i,j)=\eta*y(i)*x(j)$。由此可看出，第 j 个输入和第 i 个神经元之间的权值的变化量与输入 x(j)和输出 y(i)的乘积成正比。该函数的调用格式为

[dW,NLS]=learnh(W,X,Z,N,A,T,E,gW,gA,D,LP,LS)

该函数的用法和各个参数的定义同函数 learnk( )，如

```
>>X=rand(3,2);A=rand(3,2);w=rand(3,3);lp.lr=0.5;
>>dw=learnh(w,X,[],[],A,[],[],[],[],[],lp,[])
```

其结果显示为

```
dw =
    0.2753    0.2998    0.2730
    0.1407    0.1434    0.1438
    0.2021    0.2289    0.1966
```

16. 衰减的 Hebb 权值学习规则函数 learnhd( )

原始的 Hebb 学习规则对权值矩阵的取值未做任何限制，因而学习后权值可取任意值。为了克服这一弊病，在 Hebb 学习规则的基础上增加一个衰减项，即$\Delta w(i,j)=\eta*y(i)*x(j)-dr*w(i,j)$。衰减项的加入能够增加网络学习的“记忆”功能，并且能有效地对权值加以限

制，衰减系数 dr 的取值应该在[0,1]之间。当 dr 取为 0 时，就变成原始的 Hebb 学习规则，网络学习不具备"记忆"功能；当 dr 取为 1 时，网络学习结束后权值取值很小，不过网络能"记忆"前几个循环中学习的内容。这种改进算法可利用衰减的 Hebb 权值学习规则函数 learnhd( ) 来实现。该函数的调用格式为：

[dW,NLS]=learnhd(W,X,Z,N,A,T,E,gW,gA,D,LP,LS)

式中，LP 为学习参数，包括学习速率 LP.lr 和衰减系数 LP.dr，默认时学习速率为 0.01，衰减系数为 0.01。

该函数的用法和其余参数的定义同函数 learnk( )，如

```
>>w=eye(3);b=-0.5*ones(3,1);X=[1 0;0 1;1 1];A=hardlim(w*X,b);
>>lp.lr=0.5;lp.dr=0.05;dw=learnhd(w,X,[],[],A,[],[],[],[],[],lp,[])
```

其结果显示为

```
dw =
    0.4500         0    0.5000
         0    0.4500    0.5000
    0.5000    0.5000    0.9500
```

在 MATLAB 工作空间的命令行中输入"help assoclr"，便可得到与 4 种关联学习算法（Hebb 学习规则、Konohen 学习规则、Instar 学习规则和 Outstar 学习规则）相关的函数，进一步利用 help 命令又能得到相关函数的详细介绍。

17. 自组织特征映射权值学习函数 learnsom( )

函数 learnsom( )是根据所给出的学习参数 LP 开始的。其正常状态学习速率 LP.order_lr 的默认值为 0.9，正常状态学习步数 LP.order_steps 的默认值为 1000，调整状态学习速率 LP.tune_lr 的默认值为 0.02，调整状态邻域距离 LP.tune_nd 的默认值为 1。在网络处于正常状态和调整状态时,学习速率和邻域尺寸都得到更新。该函数的调用格式为

[dW,NLS]=learnsom(W,X,Z,N,A,T,E,gW,gA,D,LP,LS)

该函数的用法和其余参数的定义同函数 learnk( )，如以下 MATLAB 程序

```
X=rand(2,1);A =rand(4,1);w=rand(4,2);
pos=hextop(2,2);D =linkdist(pos);
lp.order_lr=0.9;                %正常状态学习速率
lp.order_steps=1000;            %正常状态学习步数
lp.tune_lr=0.02;                %调整状态学习速率
lp.tune_nd=1;                   %调整状态邻域距离
dW=learnsom(w,X,[],[],A,[],[],[],[],D,lp,[])
```

其结果显示为

```
dW =
```

-0.9046	-0.3596
-0.3741	0.0777
-1.1249	0.6372
-0.4979	0.0382

18. 绘制自组织特征映射网络的权值矢量函数 plotsom( )

函数 plotsom(W,m)用于绘制自组织映射网络的权值图，在每个神经元的权矢量（行）相应的坐标处画一点，表示相邻神经元权值的点，根据邻阵 m 用实线连接起来，即如果 M(i,j)小于等于 1，则将神经元 i 和 j 用线连接起来。其调用格式为

plotsom(W,m)

式中，W 为权值矩阵；m 为网络邻域。

例如，对两组输入为 12 个神经元（随机产生权值）利用以下命令可得到如图 2-45 所示的权值图。

```
>>W=rands(12,2);
>>m=nbman(3,4);plotsom(W,m)
```

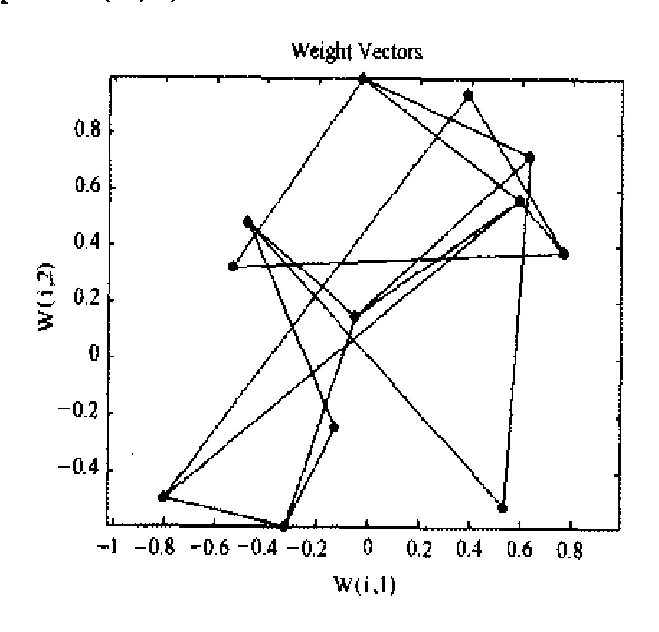


图 2-45　输入样本向量的分布

19. 利用 Kononen 规则训练自组织特征映射网络函数 trainsm( )

对自组织特征映射网络的权值初始化后，便可应用函数 trainsm( )对网络进行训练。自组织特征映射网络由一层一维或多维的神经元构成。任何时候，只有网络输入最大的神经元输出为 1，相邻的神经元输出为 0.5，其余所有神经元输出均为 0。该函数的调用格式为

W=trainsm (w,m,X,tp)

式中，w 和 W 分别为训练前后的权值矩阵；m 为网络的邻域；X 为输入向量；tp 为训练参数，其作用是设定如何进行训练，其中 tp(1)显示间隔次数，默认值为 25；tp(2)为最大循环

次数，默认值为 100；tp(3)为学习速率，默认值为 1。例如，创建一个具有 100 个元素的输入向量，构造一个排列在 3×3 栅格上由 9 个神经元组成的自组织特征映射网络可利用以下命令，返回新的权值矩阵如图 2-46 所示。

```
>>X=rand(2,100);
>>w=initsm(X,9);m=nbman(3,3);
>>W=trainsm(w,m,X,[20,400])
```

20. 自组织特征映射网络仿真函数 simusm( )

自组织特征映射网络由分布在一维或多维空间的神经元组成，在任何给定的时间，只有网络输入最大的神经元输出为 1，与获胜神经元相邻的神经元输出为 0.5，其余神经元输出均为 0。该函数的调用格式为

Y=simusm(X,w,m,n)

式中，w 为竞争层权值矩阵；X 为输入矢量；m 为网络的邻域；n 为网络邻元的大小，默认为 1；Y 为网络输出矩阵，如

```
>>w=initsm([0 2;-5 5],4);m=nbman(1,1);Y=simusm ([2;4],w,m)
```

其结果显示为

```
Y =
      (1,1)   1
```

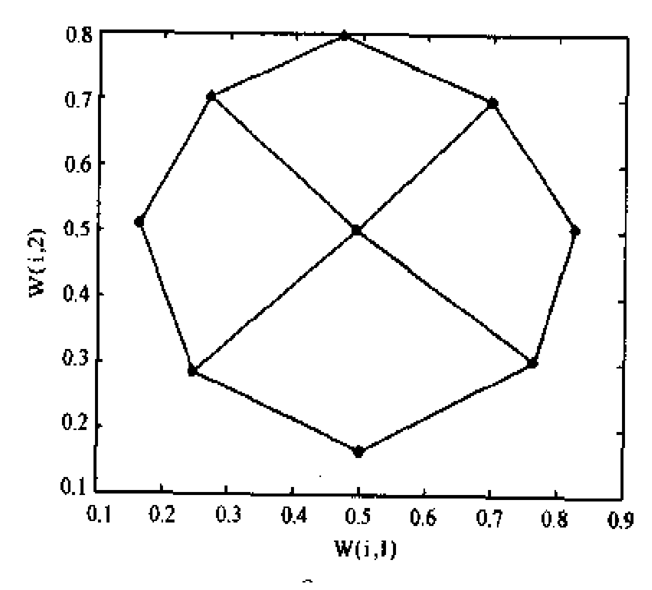


图 2-46　权值矩阵

例 2-20　建立一个具有 30 个神经元的二维自组织特征映射神经网络来对 1000 个二维随机输入向量分类，并对其进行训练和仿真。

解：MATLAB 程序如下：

```
%随机生成 1000 个二维向量作为样本，并绘制其分布
```

```
X=rands(2,1000);
plot(X(1,:),X(2,:),'+r');xlabel('X(1)') ;ylabel('X(2)')
%建立一个 5×6 结构（30 个神经元）的二维自组织映射神经网络，得到初始权值
w=initsm(X,30);m=nbman(5,6);
figure;plotsom(w,m)                %绘制初始权值的分布图
%训练神经网络，并绘制训练后权值的分布图
df=20;                             %显示间隔
me=100;                            %训练步数
lr=0.1;                            %学习速率
tp=[df me lr];w=trainsm(w,m,X,tp)
%对于训练好的网络进行测试与使用
X1=[0.5;0.3];Y=simusm(X1,w,m)
```

利用以上程序可得到如下结果，如图 2-47 和图 2-48 所示。

```
y =
   (21,1)      0.5000
   (26,1)      0.5000
   (27,1)      1.0000
   (28,1)      0.5000
```

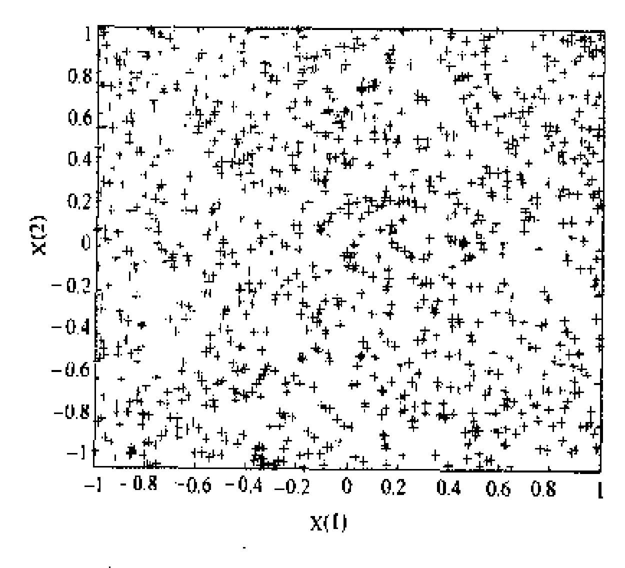


图 2-47　1000 个二维样本向量的分布

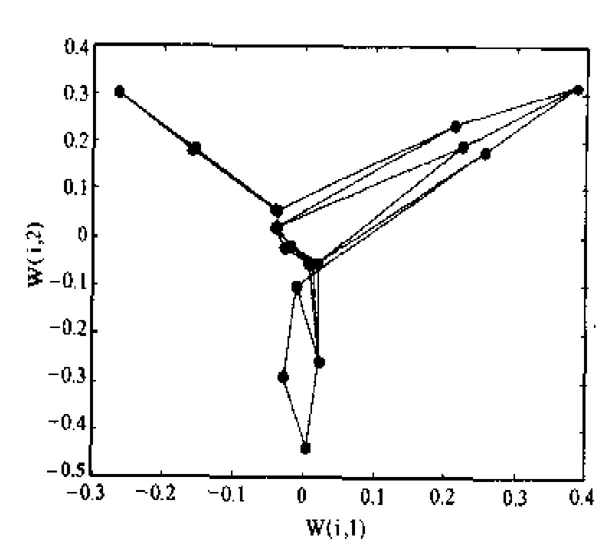


图 2-48　训练后网络权值的分布

21. 创建一个自组织特征映射网络函数 newsom( )

利用 newsom( )函数可建立一个自组织特征映射网络。其调用格式为

net=newsom (Xr,[d1,d2,…,di])

式中，Xr 为一个 $R\times2$ 维的输入矩阵；R 为输入向量的个数，它决定了输入向量的最小值和

最大值的取值范围；[d1,d2,…,di]为自组织特征映射网络维数；net 为生成的自组织特征映神经网络。例如，建立一个输入向量分布在一个二维空间，其变化范围分别为[0 2]和[0 1]，网络结构为 3×5 (15 个神经元)的二维自组织特征映射神经网络，可利用命令

```
>>net=newsom([0 2;0 1],[3 5]);
>>plotsom(net.layers{1}.positions)
```

其执行结果如图 2-49 所示。

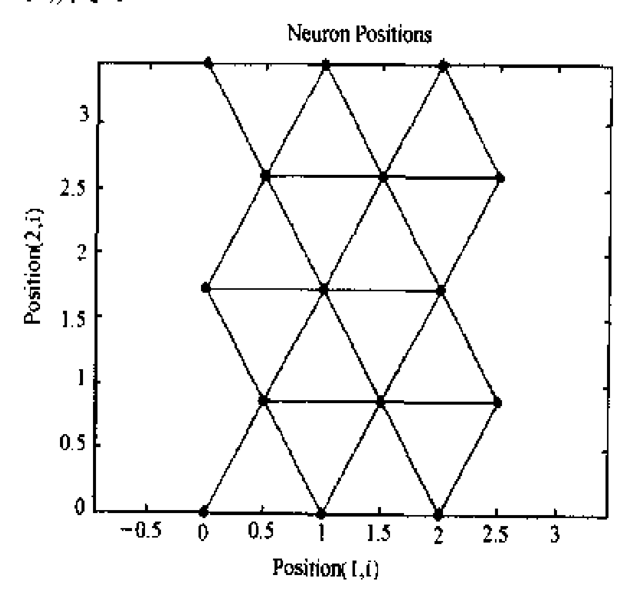


图 2-49　二维自组织特征映射神经网络

例 2-21　利用 newsom( )函数同样可以建立一个具有 30 个神经元的二维自组织特征映射神经网络来对 1000 个二维随机输入向量分类，并对其进行训练和仿真。

解：MATLAB 程序如下：

```
%随机生成 1000 个二维向量作为样本，并绘制其分布
X=rands(2,1000);plot(X(1,:),X(2,:),'+r');xlabel('X(1)') ;ylabel('X(2)')
%建立一个 5×6 结构(30 个神经元)的二维自组织映射神经网络，得到初始权值
net=newsom([0 1;0 1],[5 6]);
w =net.iw{1,1};                                    %初始权值
plotsom(w,net.layers{1}.distances)                 %绘制初始权值的分布图
%分别对不同的步长训练网络，并绘制权值分布图
for i=10:30:100
net.trainParam.epochs=i;                           %最大训练步数
net=train(net,X);
figure;plotsom(net.iw{1,1},net.layers{1}.distances)  %绘制权值
end
%对于训练好的网络进行测试与使用
```

```
X1=[0.5;0.3];y=sim(net,X1)
```

利用以上程序可得到如下结果，如图 2-50～图 2-53 所示。

```
y =
   (24,1)        1
```

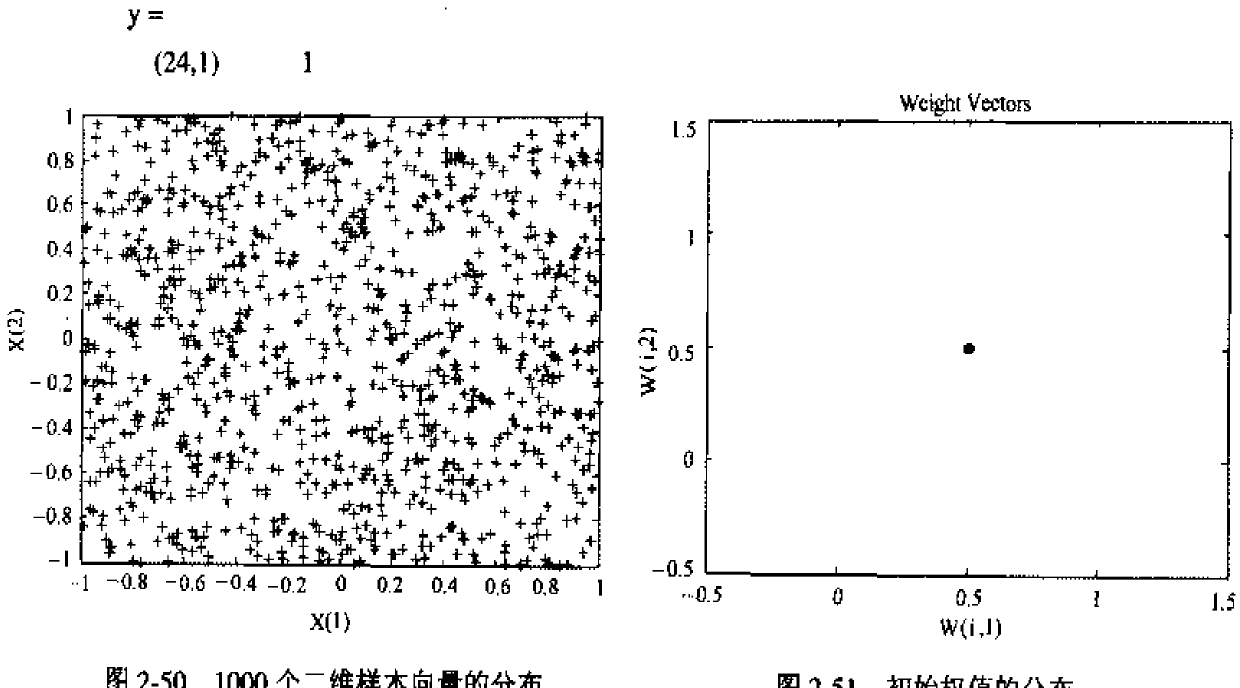


图 2-50　1000 个二维样本向量的分布

图 2-51　初始权值的分布

图 2-50 显示了这些二维随机向量的分布情况；图 2-51 显示了初始权值的分布情况；图 2-52 显示了训练次数为 10 时的权值分布情况；图 2-53 显示了训练次数为 100 时的权值分布情况。由图可知，神经元经过 10 步以后，神经元就已经自组织地分布了，每个神经元就开始能够区分输入空间中的不同区域。随着训练步数的增加，神经元的权值分布得更加合理，但当步数达到一定数目以后，这种改变就非常不明显了，如 70 步与 100 步。

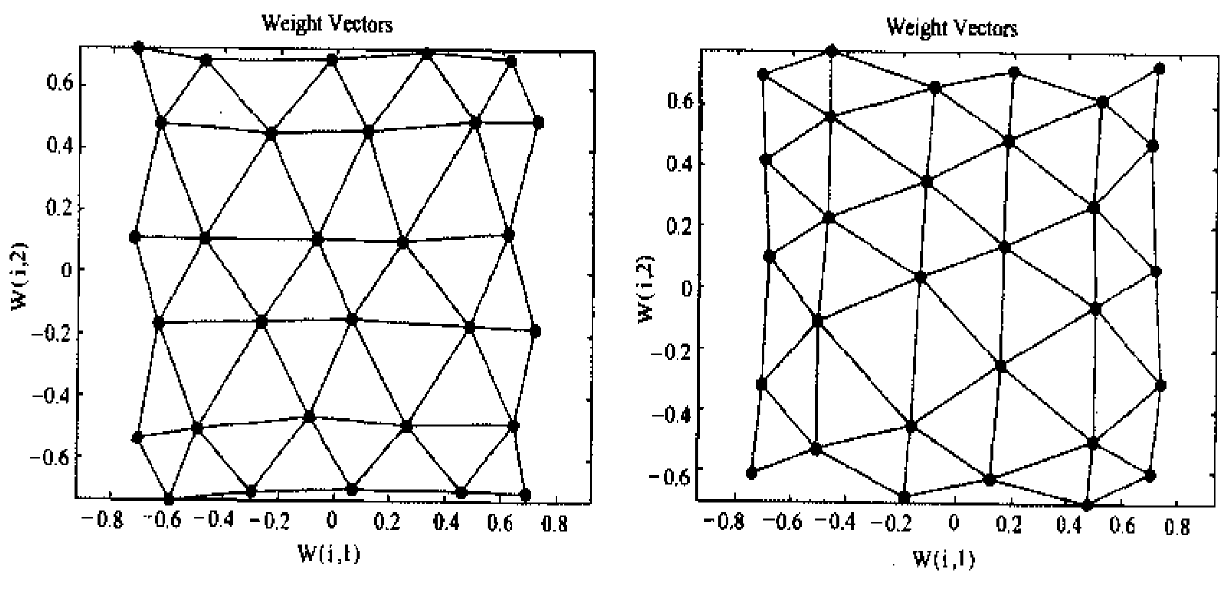


图 2-52　训练 10 步时的权值分布

图 2-53　训练 100 步时的权值分布

22. 欧氏距离权值函数 dist( )

大多数神经元网络的输入可通过表达式 N=w*X+b 来计算。其中，w，b 分别为权矢量和偏差矢量。但有一些神经元的输入可由函数 dist( )来计算。dist( )函数是一个欧氏(Euclidean) 距离权值函数。它对输入进行加权，得到被加权的输入。一般，两个向量 x 和 y 之间的欧氏(Euclidean) 距离 d 定义为 d=sum((x-y).^2).^0.5。函数 dist( )的调用格式为

d=dist(W,X)

或 d=dist(pos)

式中，W 为 $S\times R$ 权值矩阵；X 为 $R\times Q$ 输入矩阵；d 为 $S\times Q$ 的输出距离矩阵，其中含有 W 的权值（行）矢量和 X 的输入（列）矢量的矢量距离。d=disk(pos) 函数也可以作为一个阶层距离函数，用于查找某一层神经网络中的所有神经元之间的欧氏距离，且函数也返回一个距离矩阵。例如，任意给定输入和权值，利用 dist( )函数计算欧氏距离。

```
>>w=[1 2 3];X=[1;2.1;0.9];d=dist(w,X)
```

其结果显示为

```
d =

    2.1024
```

23. Manhattan 距离权值函数 mandist( )

mandist( )函数是一个 Manhattan 距离权值函数。它对输入进行加权，得到被加权的输入。一般，两个向量 x 和 y 之间的 Manhattan 距离 d 定义为 d=sum(abs(x−y))。其调用格式为

d=mandist(w,X)

或 d=mandist(pos)

式中，w 是权值函数；X 为一个输入矩阵；d 为距离矩阵。d=mandisk(pos) 函数也可以作为一个阶层距离函数，用于查找某一层神经网络中的所有神经元之间的 Manhattan 距离，函数也返回一个距离矩阵。例如，任意给定输入和权值，利用 mandist( )函数计算 Manhattan 距离的 MATLAB 命令如下：

```
>>w=rand(4,3);X=rand(3,1);d=mandist(w,X)
```

24. link 距离权值函数 linkdisk( )

linkdisk( )函数是一个阶层距离函数，用于查找某一层神经网络中的所有神经元之间的 Link 距离，函数也返回一个距离矩阵。其调用格式为

d=linkdisk(Pos)

式中，Pos 为一个矩阵；d 为距离矩阵。

例如，分布在三维空间里的 10 个神经元任意定义一个矩阵，可以利用 linkdisk( )函数查找其距离的 MATLAB 命令如下：

```
>>Pos=rand(3,10);d=linkdist(Pos)
```

其结果显示为

```
d =
     0     1     1     1     1     1     2     2     1     1
     1     0     1     1     1     1     1     1     1     1
     1     1     0     1     2     1     2     2     1     1
     1     1     1     0     1     1     1     1     1     1
     1     1     2     1     0     1     1     1     1     1
     1     1     1     1     1     0     1     1     2     1
     2     1     2     1     1     1     0     1     1     1
     2     1     2     1     1     1     1     0     1     1
     1     1     1     1     1     2     1     1     0     1
     1     1     1     1     1     1     1     1     1     0
```

25. 中点权值初始化函数 midpoint( )

如果竞争层和自组织网络的初始权值选择在输入空间的中间区，则利用该函数初始化权值会更加有效。该函数的调用格式为

w=midpoint(S,Xr)

式中，Xr 为一个 $R\times 2$ 维的输入矩阵，它决定了输入向量的最小值和最大值的取值范围；R 为输入向量的个数；S 为神经元数；w 为权值。例如，利用函数 midpoint( ) 初始化权值的 MATLAB 命令如下：

```
>>Xr=[0 1;-2 2];w=midpoint(3,Xr)
```

其结果显示为

```
w =
    0.5000         0
    0.5000         0
    0.5000         0
```

26. 对输入矢量进行加权函数 negdist( )

函数 negdist( )的调用格式为

d=negdist(w,X)

式中，w 为 S×R 维权值函数；X 为一个 R×Q 维输入矩阵；d 为 S×R 维负矢量距离矩阵，即 d=−sqrt(sum(w−X)^2)，如以下 MATLAB 命令：

```
>>w=rand(4,3);X=rand(3,1);d=negdist(w,X)
```

其结果显示为

```
d =
    -0.9578
```

−1.1555
−0.3348
−0.4025

2.6　学习向量量化（LVQ）神经网络工具箱函数

学习向量量化（LVQ）神经网络是在监督状态下对竞争层进行训练的一种学习算法。LVQ 神经网络一般有两层：第一层是竞争层；第二层是将竞争层的分类结果传递到用户定义的目标分类上。竞争层自动学习对输入向量进行分类。但是，竞争层进行的分类只与输入向量之间的距离有关。如果两个输入向量非常近，那么竞争层就很有可能将它们归到一类。LVQ 神经网络还可以通过学习，将输入向量中与目标向量相近的向量分离出来。MATLAB 神经网络工具箱中提供了大量的与 LVQ 神经网络相关的工具箱函数。在 MATLAB 工作空间的命令行输入"help lvq"，便可得到与 LVQ 神经网络相关的函数，进一步利用 help 命令又能得到相关函数的详细介绍。表 2-6 列出了 LVQ 的重要函数和基本功能。

表 2-6　LVQ 的重要函数和基本功能

函 数 名	功 能
initlvq( )	初始化 LVQ 神经网络
trainlvq( )	训练 LVQ 神经网络
simulvq( )	仿真 LVQ 神经网络
newlvq( )	建立一个 LVQ 神经网络函数
learnlvq( )	LVQ 神经网络学习函数
plotvec( )	用不同的颜色画矢量函数

1. LVQ 神经网络的初始化函数 initlvq( )

利用 initlvq( )函数可建立一个两层（一个竞争层和一个输出层）LVQ 神经网络。其调用格式为

[W1,W2]=initlvq(X,S1,S2)

或

[W1,W2]=initlvq(X,S1,T)

式中，X 为输入矢量矩阵，X 中的每一行应包含网络输入的所有可能的最小值和最大值，这样的权值才能正确地初始化；S1 为隐含竞争层的神经元数；S2 为线性输出层的神经元数，S2 也可用目标向量 T 来代替，此时输出神经元数 S2 根据 T 中的行数来设置；W1 为网络竞争层的初始权值；W2 为网络输出层的初始权值。另外，T 中的矢量反映了类所期望的分布。换句话说，如果期望网络的 25%的输入矢量出现在 j 类中，则在 T 中的 25%的矢量除在元素 j 上有一个 1 之外，其余均为 0。

2. LVQ 神经网络的训练函数 trainlvq( )

一个 LVQ 神经网络由一个竞争层和一个线性输出层组成。竞争层的神经元将输入矢量分成组，然后由线性输出层组合到期望的类别中。任何时候，只有一个线性输出神经元具有非零输出 1。这样，如果网络有 5 个输出神经元，则它能把输入矢量分成 5 类。训练 LVQ 神经网络函数 trainlvq( )的调用格式为

[W1,W2]= trainlvq(w1,w2,X,T,tp)

式中，w1 和 w2 为网络的初始权值；X 为网络的输入向量矩阵；T 表示网络的目标向量；tp=[disp_freq max_cycle lr]是训练控制参数，包括显示频率（默认值为 25）、最大迭代次数（默认值为 100）和学习速率（默认值为 0.01）；W1 和 W2 为网络训练后的权值。

3. LVQ 神经网络的仿真函数 simulvq( )

该函数的调用格式为

[y1,y2]=simulvq(X,w1,w2)

式中，X 为网络的输入向量矩阵；w1,w2 分别为网络竞争层和线性输出层的权值；y1,y2 分别为网络竞争层和线性输出层的输出向量。

例 2-22 设计一个学习向量量化网络，根据目标，将输入向量进行模式分类。

解：一个 LVQ 网络由一个隐含竞争层和一个输出层组成。竞争层的神经元将输入向量分成组，然后由线性输出层组合到期望的类别中。任何时候，只有一个线性输出神经元具有非零输出 1。MATLAB 程序如下：

```
%指定输入二维向量及其类别并将这些类别转换成学习向量量化网络使用的目标向量，且绘制这些输入向量
X=[-3 -2 -2 0 0 0 0 2 2 3;0 1 -1 2 1 -1 -2 1 -1 0];
C=[1 1 1 2 2 2 2 1 1 1];
T=ind2vec(C); plotvec(X,C);
%建立一个学习向量量化网络
S1=4;
[W1,W2]=initlvq(X,S1,T)
figure;plotvec(X,C); hold on;
xlabel('X(1),w(1)') ;ylabel('X(2),w(2)');
hold off;
%训练网络，并绘制权值分布图
disp_freq=20;
max_cycle=500;
lr=0.05;
tp=[disp_freq  max_cycle  lr];
```

```
[W1,W2]=trainlvq(W1,W2,X,T,tp);
%对于训练好的网络进行测试与使用
X1=[0.8;0.3];
y2=simulvq(X1,W1,W2)
y=vcc2ind(y2)
yy=full(y2)              %y2 的全矩阵形式
```

利用以上程序可得到图 2-54 和图 2-55 及如下结果。

```
y2 =
     (1,1)   1
y =
     1
yy =
     1
     0
```

结果 y2 是用稀疏矩阵形式（1 行第 1 列的元素为 1）表示的。它表明第一个神经元有输出，即输入向量 X1 属于 1 类，与事实相吻合。学习向量量化网络能够对任意输入向量进行分类，不管它们是不是线性可分的。这一点比感知机神经网络要优越得多。但需要注意，在竞争层必须要有足够多的神经元，使得每一个类有足够多的竞争神经元。

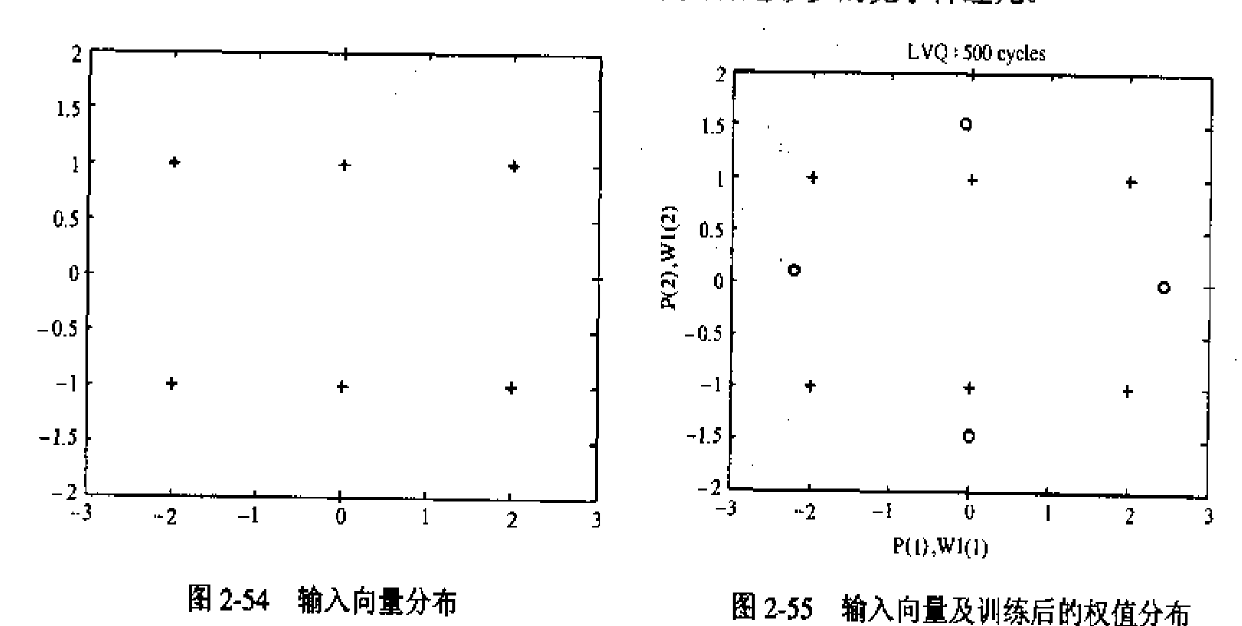


图 2-54　输入向量分布　　　图 2-55　输入向量及训练后的权值分布

4. 建立一个向量量化神经网络函数 newlvq( )

利用 newlvq( )函数可建立一个向量量化（LVQ）神经网络。其调用格式为

net = newlvq(Xr,S1,Pc,Lr, Lf)

式中，Xr 为一个 $R\times2$ 维的输入矩阵，它决定了输入向量的最小值和最大值的取值范围；R

为输入向量的个数；S1 表示隐含层神经元的数目；Pc 表示在第二层的权值中列所属类别的百分比；Lr 表示学习速率，默认值为 0.01；Lf 表示学习函数，默认值为'learnlvq'；net 为生成的自组织竞争网络。例如，建立一个隐含层有 4 个神经元，在第二层的权值中，有 60%的列第一行的值为 1，40%的列第二行的值为 1，即 60%的列属于第一类，40%的列属于第二类，学习速率为 0.1 的向量量化神经网络。MATLAB 命令如下：

```
>>X=[rand(1,400)*2; rand(1,400)];
>>net=newlvq(minmax(X),4,[0.6 0.4],0.1);w=net.iw{1}
```

其结果显示为

```
w =
      1.0050    0.5003
      1.0050    0.5003
      1.0050    0.5003
      1.0050    0.5003
```

5. LVQ 神经网络学习函数 learnlvq( )

对 LVQ 网络进行初始化后，即可对其进行学习。LVQ 网络的学习规则是由 Kohonen 学习规则（learnk）发展而来的。在 LVQ 网络中，竞争层的目标向量 T1 可由网络的目标向量 T 和线性层的权值 w1 得到，即 T1=w1'*T。该函数的调用格式为

dw1=learnlvq(w1,X,y1,T1,lr)

式中，w1 为竞争层权值；X 为输入向量矩阵；y1 表示竞争层的输出向量；T1 表示竞争层的目标向量；dw1 为竞争层权值变化阵；lr 为学习速率。

6. 用不同的颜色画矢量函数 plotvec( )

函数 plotvec(X,c,m)包含一个列矢量矩阵 X、标记颜色的行矢量 c 及一个图形标志 m。X 的每个列矢量用图形标志画图。每列矢量 X(:,i)的数据颜色为 c(i)。如果 m 默认，则用默认图形标志“+”，如

```
>>x=[0 1;-1 2];c=[1 2];plotvec(x,c)
```

例 2-23 利用函数 learnlvq( )设计一个学习向量量化网络，根据目标，将输入向量进行模式分类。

解：MATLAB 程序如下：

```
%指定输入二维向量及其类别并将这些类别转换成学习向量量化网络使用的目标向量，且绘制这些输入向量
X=[-3 -2 -2 0 0 0 0 2 2 3;0 1 -1 2 1 -1 -2 1 -1 0];
C=[1 1 1 2 2 2 2 1 1 1];
T=ind2vec(C); plotvec(X,C);
%建立一个学习向量量化网络
```

```
net=newlvq(minmax(X),4,[0.6 0.4],0.1)
w=net.iw{1};                                    %初始权值
figure;plotvec(X,C); hold on;
plot(w(1,1),w(1,2),'ow');                       %初始权值分布图
xlabel('X(1),w(1)') ;ylabel('X(2),w(2)');
hold off;
%训练网络，并绘制权值分布图
net.trainParam.epochs=150;                      %最大训练步数
net.trainParam.show=Inf;
net=train(net,X,T);
figure;plotvec(X,C);hold on;
plotvec(net.iw{1},vec2ind(net.lw{2}),'o')
%对于训练好的网络进行测试与使用
X1=[0.8;0.3];
y1=sim(net,X1)
y=vec2ind(y1)
```

利用以上程序可得到图 2-56 和图 2-57 及如下结果。

```
y1 =
     1
     0
y =
     1
```

结果显示输入向量 X1 属于 1 类，与事实相吻合。

图 2-56　输入向量及初始权值分布

图 2-57　输入向量及训练后的权值分布

2.7 Elman神经网络工具箱函数

Elman神经网络通常由输入层、隐含层和输出层构成。它存在从隐含层的输出到隐含层输入的反馈。这种反馈连接的结构使得其被训练后不仅能够识别和产生空域模式，还能够识别和产生时域模式。MATLAB神经网络工具箱中提供了建立、训练和仿真Elman神经网络的有关函数。在MATLAB工作空间的命令行输入“help elman”，便可得到与Elman神经网络相关的函数，进一步利用help命令又能得到相关函数的详细介绍。表2-7列出了Elman神经网络的重要函数和基本功能。

表2-7 Elman神经网络的重要函数和基本功能

函数名	功能
initelm( )	对Elman神经网络进行初始化
trainelm( )	训练Elman神经网络的权值和偏值
simuelm( )	对Elman神经网络进行仿真
newelm( )	生成一个Elman神经网络

1. 初始化Elman神经网络函数initelm( )

Elman神经网络是一种回归网络。它包含一个Tansig隐含层和一个Purelin输出层。Tansig隐含层接收网络输入和自身的反馈。Purelin层从Tansig隐含层得到输入。由于Elman神经网络是Sigmoid/Linear网络，能够表达含有有限个不连续点的函数。因为它们有一个反馈连接，所以它们也能够用来识别和产生时间模式与空间模式。函数initelm( )的调用格式为

[W1,b1,W2,b2]=initelm (X,S1,S2)

或 [W1,b1,W2,b2]=initelm (X,S1,T)

式中，X为一个输入向量矩阵，值得注意的是，X中的每一行都要包含期望的网络输入最小值和最大值，以便权值和偏值能合理地初始化；S1为隐含层Tansig神经元的数目；S2为输出层Purelin神经元的数目；W1和b1为隐含层的权值和偏值；W2和b2为输出层的权值和偏值。另外，输出层神经元的个数S2可以用对应的目标向量T来代替，此时输出神经元数根据T中的行数来设置。例如，设计一个隐含层有3个神经元，传输函数为Tansig，输出层有两个神经元，传输函数为Purelin的两层Elman神经网络可利用以下命令，即

```
>>X=[sin(0:100);cos([0:100]*2)];[W1,b1,W2,b2]=initelm(X,3,2);
```

2. 训练Elman神经网络函数trainelm( )

该函数的调用格式为

[W1,B1,W2,B2,te,tr]=trainelm(w1,b1,w2 ,b2,X,T,tp)

式中，w1,W1和b1, B1分别为训练前后Tansig隐含层的权值矩阵和偏值向量；w2,W2和b2,

B2 分别为训练前后 Purelin 输出层的权值矩阵和偏值向量；te 为实际训练次数；tr 为网络训练误差平方和的行向量；X 为输入向量；T 为目标向量；tp 是训练控制参数，其作用是设定如何进行训练，其中 tp(1)显示间隔次数，默认值为 5；tp(2)为最大循环次数，默认值为 500；tp(3)为目标误差平方和，默认值为 0.01；tp(4)为学习速率，默认值为 0.001；tp(5)为学习速率增长系数，默认值为 1.05；tp(6)为学习速率减小系数，默认值为 0.7；tp(7)为动量常数，默认值为 0.95；tp(8)为最大误差率，默认值为 1.04。

3. 仿真 Elman 神经网络函数 simuelm( )

该函数的调用格式为

y=simuelm (X,w1,b1,w2,b2,A1)

式中，X 为输入向量；w1,w2 为权值矩阵；b1,b2 为偏值向量；A1 为隐含层的初始输出，可省略；y 为网络输出。

例 2-24　设计一个 Elman 神经网络，用于对输入波形进行振幅检测。

解：MATLAB 程序如下：

```
%定义输入信号及目标信号
Time=1:80;
X1=sin(1:20);X2=2*sin(1:20);
t1=ones(1,20);t2=2*ones(1,20);
X=[X1 X2 X1 X2];T=[t1 t2 t1 t2];
%X=con2seq(P);T=con2seq(t);                    %将矩阵信号转换为序列信号
%绘制输入信号及目标信号曲线
figure;plot(Time,X,'--',Time,T); xlabel('t');ylabel('X,T');
%建立网络，并得权值和偏值
S1=10;[w1,b1,w2,b2]=initelm(X,S1,T);
%训练网络
df=10;                                         %显示间隔
me=5000;                                       %训练次数
tp=[df me];
figure;[w1,b1,w2,b2]=trainelm(w1,b1,w2,b2,X,T,tp)
%测试网络的性能
y=simuelm(X,w1,b1,w2,b2);
%绘制输出信号及目标信号曲线
figure;plot(Time,y,'--',Time,T); xlabel('t');ylabel('y,T');
```

执行结果可得到如图 2-58 和图 2-59 所示的曲线。

从如图 2-59 所示的曲线可以看出，输出信号在目标信号的两侧有小幅度的振荡，但能跟着输入信号的振幅变化而变化，网络能较好地检测出输入信号的振幅。

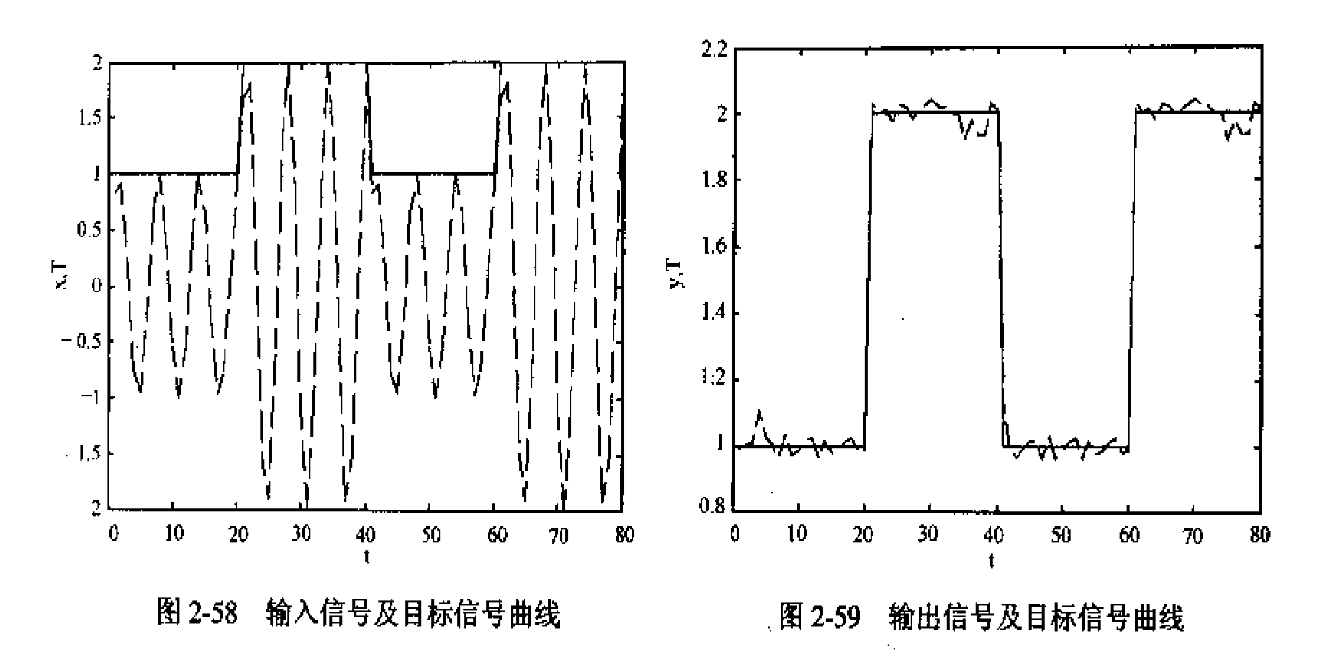


图 2-58 输入信号及目标信号曲线

图 2-59 输出信号及目标信号曲线

4. 建立网络函数 newelm( )

利用 newelm( )函数可建立一个 Elman 神经网络。其调用格式为

net=newelm(Xr,[S1 S2 … Sn],{TF1, TF1,…, TFn},BTF,BLF,PF)

式中，Xr 为一个 $R\times2$ 输入矩阵，它决定了输入向量的最小值和最大值的取值范围；Si 表示网络的第 i 层神经元数(在 MATLAB 神经网络工具箱中隐含层为第一层，其后依次为第二、三层)；TFi 表示第 i 层的传输函数；BTF 表示反向传播网络的训练函数，默认时为'tansig'；BLF 表示网络的反向传播权值或偏值学习函数，默认时为'learngdm'；PF 表示性能分析函数，默认时为'mse'；net 为生成的 Elman 神经网络。

例 2-25 利用 newelm( )函数设计一个 Elman 神经网络，用于对输入波形进行振幅检测。

解：MATLAB 程序如下：

```
%定义输入信号及目标信号
Time=1:80;X1=sin(1:20);X2=2*sin(1:20);t1=ones(1,20);t2=2*ones(1,20);
P=[X1 X2 X1 X2];t=[t1 t2 t1 t2];
X=con2seq(P);T=con2seq(t);                    %将矩阵信号转换为序列信号
%绘制输入信号及目标信号曲线
figure;plot(Time,cat(2,X{:}),'--',Time,cat(2,T{:}));
%建立网络，并得到权值和偏值
[R,N]=size(X);[S2,N]=size(T);S1=10;
net=newelm([-2 2],[S1 S2],{'tansig','purelin'},'traingdx');
xlabel('t');ylabel('X,T');
%训练网络
net.trainParam.epochs=5000;                   %训练时间
```

```
[net,tr]=train(net,X,T);
%测试网络的性能
y=sim(net,X);
%绘制输入信号及目标信号曲线
figure;plot(Time,cat(2,y{:}),Time,cat(2,T{:}),'--');
xlabel('t');ylabel('y,T');
```

执行结果可得到如图 2-60 和图 2-61 所示的曲线。

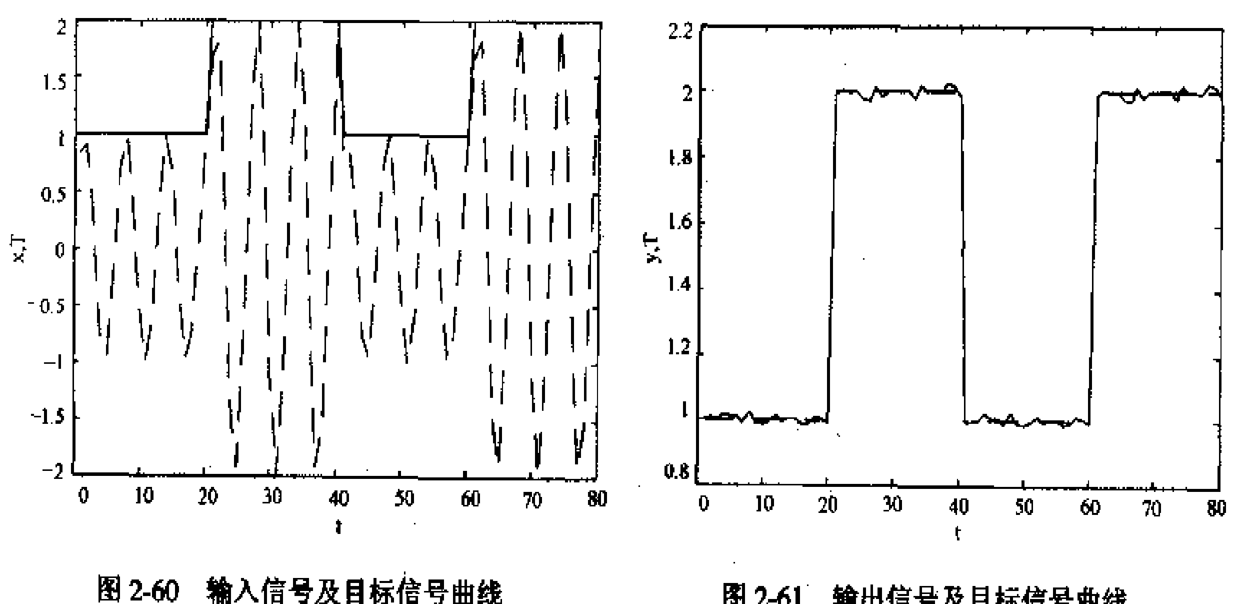


图 2-60　输入信号及目标信号曲线　　　　图 2-61　输出信号及目标信号曲线

从如图 2-61 所示的曲线可以看出，输出信号在目标信号的两侧也有小幅度的振荡，但与如图 2-59 所示相比，其振荡幅度有所减小。

2.8　Hopfield 神经网络工具箱函数

Hopfield 神经网络常用于存储一个或者多个稳定的目标向量。当向网络提供输入向量时，存储在网络中的接近于输入的目标向量就被唤醒。Hopfield 神经网络设计的目标就是使得网络存储一些特定的平衡点，当给定网络一个初始条件时，网络最后会在这样的点上停下来。由于输出又被反馈到输入，所以一旦网络开始运行，整个网络就是递归的。MATLAB 神经网络工具箱提供了建立、训练和仿真 Hopfield 神经网络的有关函数。在 MATLAB 工作空间的命令行输入“help hopfield”，便可得到与 Hopfield 神经网络相关的函数，进一步利用 help 命令又能得到相关函数的详细介绍。表 2-8 列出了 Hopfield 网络的重要函数和基本功能。

表 2-8 Hopfield 网络的重要函数和基本功能

函数名	功能
satlin( )	饱和线性传输函数
satlins( )	饱和对称线性传输函数
newhop( )	生成一个 Hopfield 回归网络
solvehop( )	设计一个 Hopfield 回归网络
simuhop( )	仿真一个 Hopfield 回归网络

1. 饱和线性传输函数 satlin( )

对饱和线性传输函数，如果输入大于 0 且小于 1，则返回其输入值；如果输入小于 0，则返回 0；如果输入大于 1，则返回 1。因此，satlin 神经元在一定的输入区域内像线性神经元，超出此区域，则像硬特性神经元。该函数的调用格式为

a=satlin(N)

或

a=satlin(Z,b)

a=satlin(P)

函数 satlin (N)在给定网络的输入矢量矩阵 N 时，返回该层的输出矢量矩阵 a。当 N 中的元素介于 0～1 之间时，其输出等于输入，在输入值小于 0 时返回 0，大于 1 时返回 1。函数 satlin(Z,b)用于矢量是成批处理且存在偏差的情况下，此时的偏差 b 和加权输入矩阵 Z 是分开传输的。偏差矢量 b 加到 Z 的每个矢量中形成网络输入矩阵，然后用上述方法转换成输出矩阵。返回的元素 a 是 1 还是 0，取决于网络输入矩阵中的元素；函数 satlin(P)包含传输函数的特性名并返回问题中的特性，如下的特性可从任何传输函数中获得：

（1）delta——与传输函数相关的 delta 函数；

（2）init——传输函数的标准初始化函数；

（3）name——传输函数的全称；

（4）output——包含有传输函数最小、最大值的二元矢量。

式中，N 为 Q 组 S 维的网络输入向量。例如，利用以下命令可得到如图 2-62 所示的饱和线性传输函数。

```
>>n=-3:0.1:3;a=satlin(n);plot(n,a)
```

2. 饱和对称线性传输函数 satlins( )

对饱和对称线性传输函数，如果输入大于-1 且小于 1 时，则返回其输入值；如果输入小于-1，则返回-1；如果输入大于 1，则返回 1。因此，satlins 神经元在一定的输入区域内也像线性神经元，超出此区域，则像硬特性神经元。该函数的调用格式同函数 satlin( )。

3. 建立网络函数 newhop( )

Hopfield 神经网络经常被应用于模式的联想记忆中。Hopfield 神经网络仅有一层，其输

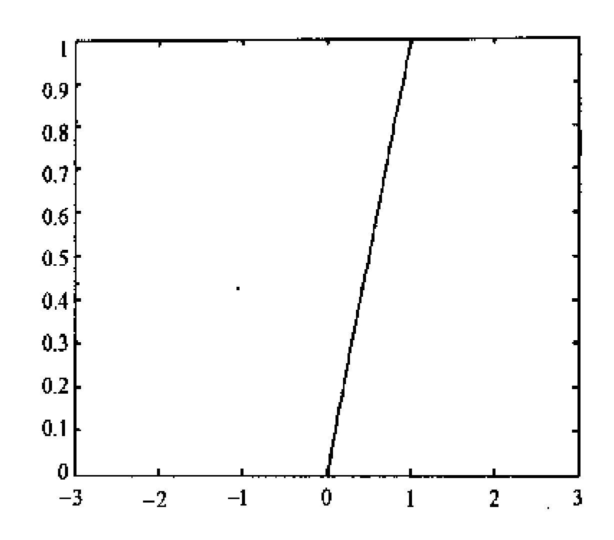


图 2-62 饱和线性传输函数

入用 netsum( )函数，权函数用 dotprod( )函数，传输函数用饱和对称线性 satlins( )函数，层中的神经元有来自它自身的连接权和偏值。该函数的调用格式为

net=newhop(T)

式中，T 为目标向量；net 为生成的神经网络。例如，利用以下命令可得到一个 Hopfield 网络的每个神经元的权值和偏值。

```
>>T=[1 -1;-1 1];net=newhop(T);
>>W=net.LW{1,1},b=net.b{1,1}
```

其结果显示为

```
W =
     0.6925    -0.4694
    -0.4694     0.6925
b =
       0
       0
```

4. 设计网络函数 solvehop( )

Hopfield 神经网络由一系列对称饱和线性神经元组成。神经元的输出通过权矩阵反馈回输入。由任何初始输出矢量开始，网络不断修正，直到有一个稳定的输出矢量。这些稳定的输出矢量认为是由初始矢量调用所唤醒的记忆。设计 Hopfield 网络包括计算对称饱和线性层的权值和偏差，以便目标矢量是网络的稳定输出矢量。该函数的调用格式为

[w,b]=solvehop(T)

式中，T 为目标向量；w 和 b 为生成的神经网络的权值和偏值。例如，利用以下命令可得到一个 Hopfield 网络的每个神经元的权值和偏值。

```
>>T=[1 -1;-1 1]; [w,b]=solvehop(T)
```

其结果显示为

```
w =
     0.6925    -0.4694
    -0.4694     0.6925
b =
     0
     0
```

5. 网络仿真函数 simuhop( )

Hopfield 神经网络由一层饱和线性神经元组成。神经元的输出通过权矩阵与输入相连。神经元的输出分配给初始状态后，神经元由任何初始输出矢量开始，网络不断修正，直到有一个稳定的输出矢量，网络可进行任意次的迭代修正。在某些点上，网络达到稳定，新的输出等于以前的输出，最后的输出矢量可以看做是初始矢量的分类。每个 Hopfield 网络都有有限个这样稳定的输出矢量。利用 simuhop( )函数可以测试一个 Hopfield 神经网络的性能。该函数的调用格式为

[y,yy]=simuhop(T,w,b,ts)

式中，T 为初始的输出向量；w 和 b 为神经网络的权值和偏值；ts 为迭代次数；y 为输出终值；yy 为在所有仿真点的输出值矩阵。例如，利用以下命令可用目标向量 T 来测试其是否被存储在网络中，即将目标向量作为输入

```
>>T=[1 -1;-1 1]; [w,b]=solvehop(T);
>>y=simuhop(T,w,b)
```

其结果显示为

```
y =
      1    -1
     -1     1
```

所得结果显示，网络的输出正好为目标向量。

例 2-26 设计一个 Hopfield 神经网络，并检验这个网络是否稳定在这些点上。

解：MATLAB 程序如下：

```
   >>T=[-1 -1 1;1 -1 1];             %给定目标值
   >>net=newhop(T);                  %设计一个 Hopfield 神经网络
   >>[y,Pf,Af]=sim(net,3,[],T)       %检验这个网络是否稳定在这些点上
或 >>T=[-1 -1 1;1 -1 1];             %给定目标值
   >>[w,b]=solvehop(T);              %设计一个 Hopfield 神经网络
   >>y=simuhop(T,w,b)                %检验这个网络是否稳定在这些点上
```

其结果显示为

```
y =
    -1    -1     1
     1    -1     1
```

例 2-27 设计一个具有两个神经元的 Hopfield 神经网络，每个神经元有一个偏值和一个权值。

解：具有两个神经元的 Hopfield 神经网络刚好与具有两个元素的输入向量相匹配。

```
%定义存储在网络中的两个目标平衡点
T=[1 -1;-1 1];
%建立网络，并得权值和偏值
[w,b]=solvehop(T)
%使用原始平衡点仿真网络
y=simuhop(T,w,b)
%使用一个随机点仿真网络，并绘制其达到稳定点的轨迹
a=rands(2,1);
y=simuhop(a,w,b,20);
plot(T(1,:),T(2,:),'r*');axis([-1.1 1.1 -1.1 1.1]);
xlabel('T(1),a(1)');ylabel('T(2),a(2)');
record=[cell2mat({a}) cell2mat({y})];     %组合一个细胞数组矩阵到一个矩阵中
start=cell2mat({a});
hold on; plot(start(1,1),start(2,1),'bx',record(1,:),record(2,:));
%使用 10 个随机点，对网络仿真 20 步，测试输出结果，并绘制达到稳定点的轨迹
figure;color='rgbmy';
plot(T(1,:),T(2,:),'r*');axis([-1.1 1.1 -1.1 1.1]);
xlabel('T(1),a(1)');ylabel('T(2),a(2)');hold on;
for i=1:10
a=rands(2,1);
[y,yy]=simuhop(a,w,b,20);
plot(yy(1,1),yy(2,1),'kx',yy(1,:),yy(2,:),color(rem(i,5)+1));
drawnow
end;
```

利用以上程序可得到图 2-63 和图 2-64 及如下结果。

```
w =
     0.6925   -0.4694
    -0.4694    0.6925
```

```
b =
     0
     0
y =
     1    -1
    -1     1
```

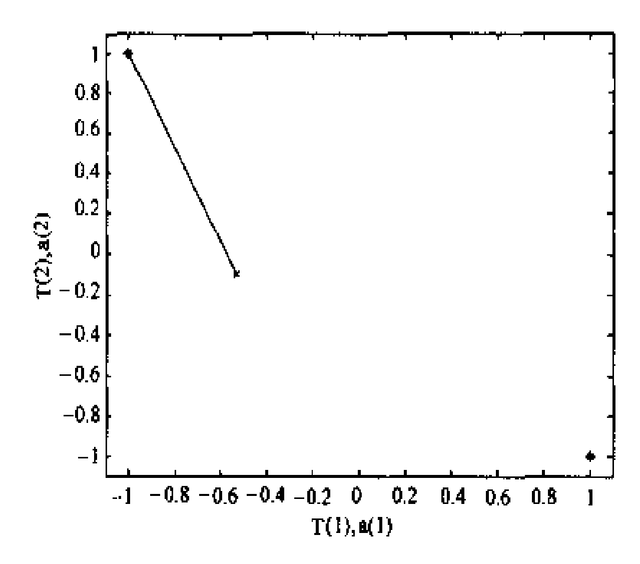


图 2-63 一个随机点趋于稳定点的轨迹

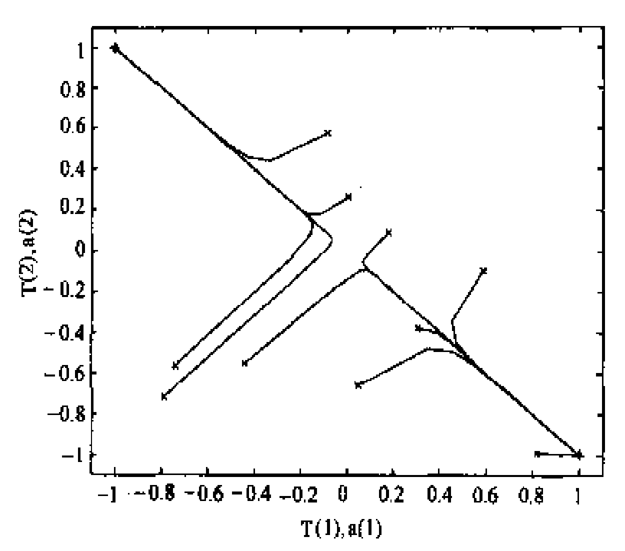


图 2-64 10 个随机点趋于稳定点的轨迹

例 2-28 设计一个具有三个神经元的 Hopfield 神经网络，每个神经元有一个偏值和一个权值。

解：具有三个神经元的 Hopfield 神经网络刚好与具有三个元素的输入向量相匹配。

```
%定义存储在网络中的两个目标平衡点
T=[1 1;-1 1;-1 -1];
%建立网络，并得权值和偏值
[w,b]=solvehop(T)
%使用原始平衡点仿真网络
y=simuhop(T,w,b)
%使用一个随机点仿真网络，并绘制其达到稳定点的轨迹
a=rands(3,1);
[y,yy]=simuhop(a,w,b,20);
figure;axis([-1 1 -1 1 -1 1]);
set(gca,'box','on'); axis manual;
hold on; plot3(T(1,:),T(2,:),T(3,:),'r*');
```

```
hold on;
plot3(yy(1,1),yy(2,1),yy(3,1),'bx',yy(1,:),yy(2,:),yy(3,:));
xlabel('T(1),a(1)'); ylabel('T(2),a(2)'); zlabel('T(3),a(3)');view([37.5 30]);
%使用 10 个随机点，对网络仿真 20 步，测试输出结果，并绘制达到稳定点的轨迹
figure;color='rgbmy';
axis([-1 1 -1 1 -1 1]);set(gca,'box','on');
xlabel('T(1),a(1)'); ylabel('T(2),a(2)');
zlabel('T(3),a(3)');view([37.5 30]);
axis manual;hold on;
plot3(T(1,:) ,T(2,:),T(3,:),'r*');
hold on;
for i=1:10
a=rands(3,1);
[y,yy]=simuhop(a,w,b,20);
plot3(yy(1,1),yy(2,1),yy(3,1),'kx',yy(1,:),yy(2,:),yy(3,:),color(rem(i,5)+1));
drawnow
end;
```

利用以上程序可得到图 2-65 和图 2-66 及如下结果。

```
w =
    0.2231         0         0
         0    1.1618         0
         0         0    0.2231
b =
    0.8546
         0
   -0.8546
y =
     1     1
    -1     1
    -1    -1
```

根据以上结果可知，网络输出 y 最终到达了稳定点。从图中可以看出，对于设计好的网络，其随机输入点最终都到达了较近的稳定点。

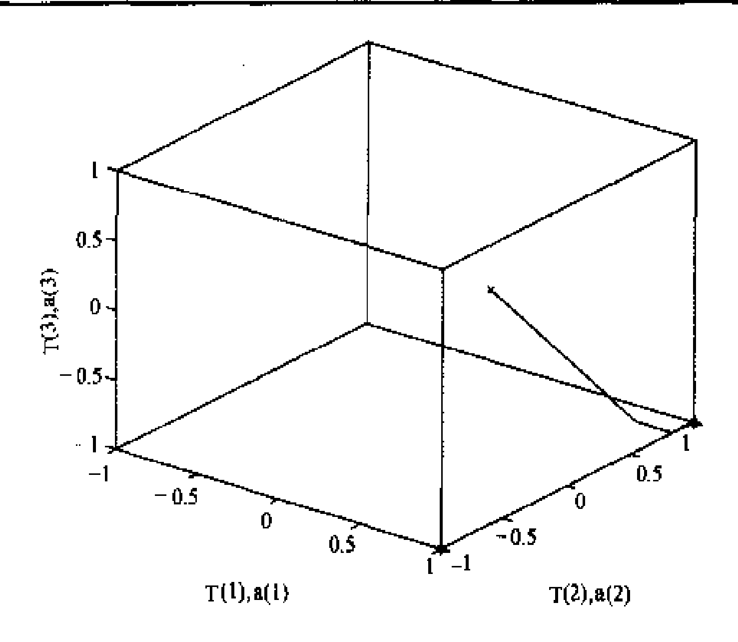


图 2-65 一个随机点趋于稳定点的轨迹

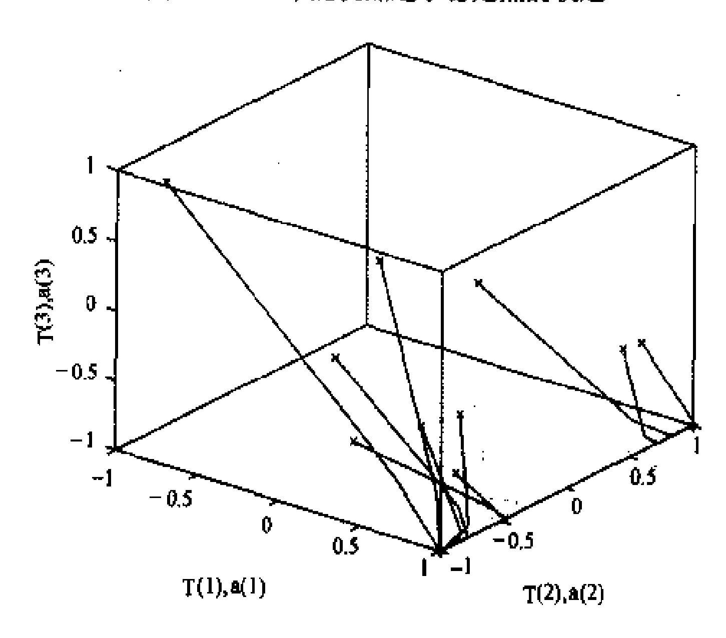


图 2-66 10 个随机点趋于稳定点的轨迹

2.9 MATLAB 神经网络工具箱的图形用户界面

前面介绍了神经网络工具箱中有关构造神经网络系统的函数。这些函数都是在 MATLAB 命令行窗口执行并显示结果的。为了进一步方便用户，神经网络工具箱提供了一个图形用户接口（GUI）的神经网络编辑器（Network/Data Manager），它可以建立网络、训练网络及仿真网络、将数据导出到 MATLAB 工作空间中、清除数据、从 MATLAB 工作空间中导入数据、变量存盘与读取。

在 MATLAB 命令窗口中，可以用以下两种方法启动神经网络编辑器（Network/Data

Manager)：

（1）在 MATLAB 的命令窗口中直接输入 nntool 命令；

（2）在 MATLAB 的 Launch Pad 窗口中，用鼠标双击神经网络系统工具箱(Neural Network Toolbox)中的 NNTool 项。

在以上两种方式启动下，神经网络编辑器的图形界面如图 2-67 所示。

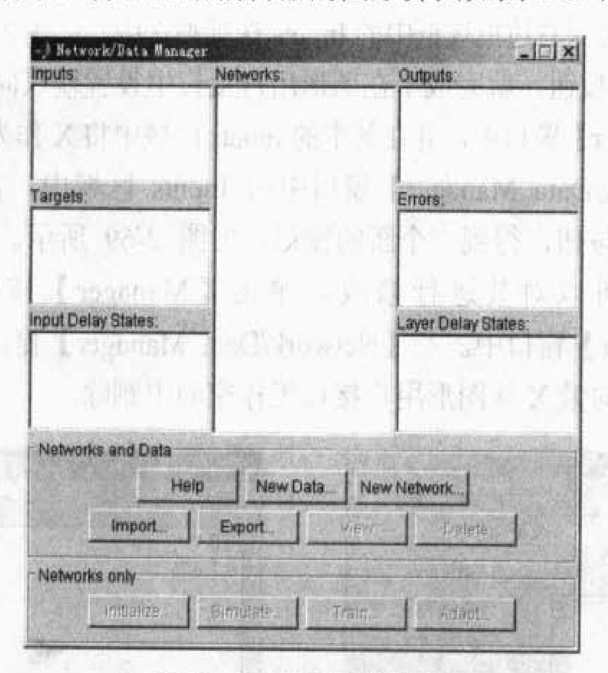


图 2-67　神经网络编辑器窗口

从图 2-67 可以看到，在窗口上半部有 7 个显示区域，在窗口下半部有 11 个按钮。其中：

Inputs 区域——显示指定的输入向量；

Targets 区域——显示指定的目标向量；

Outputs 区域——显示网络的输出值；

Errors 区域——显示误差；

Networks 区域——显示设置的网络；

Input Delay States 区域——显示设置的输入延时状态；

Layer Delay States 区域——显示设置层的延时状态。

如果对神经网络的图形用户接口不太熟悉，则可以单击【Help】按钮。这时将弹出一个新的窗口，此窗口首先介绍了如何使用图形用户接口解决问题的一般步骤，然后给出了关于这些按钮的描述，最后对每个区域各代表什么含义也进行了介绍。

下面通过一个简单的例子来说明神经网络编辑器（Network/Data Manager）的使用。

例 2-29　建立一个感知机网络，使其能够完成“或”的功能。

解：为了完成“或”函数，建立一个两输入、单输出的一个感知机网络。设输入向量

为 X=[0 0 1 1;0 1 0 1]，目标向量为 T=[0 1 1 1]。

（1）启动神经网络编辑器（Network/Data Manager）的图形界面，如图 2-67 所示。

（2）设置输入向量 X 和目标向量 T。

在如图 2-67 所示的图形界面中单击【New Data】按钮，弹出一个名为【Create New Data】的数据输入窗口。将此窗口中 Name 区域的值设置为 X，Value 区域的值设置为[0 0 1 1;0 1 0 1]，并且确保右边单选框中的 Inputs 选项被选中，如图 2-68 所示。在设置好输入向量后，单击【Create】按钮，就完成了在图形用户接口中设置输入向量的过程。此时，返回到【Network/Data Manager】窗口中，并在其中的 Inputs 区域中将 X 作为输入显示出来。

此时，在【Network/Data Manager】窗口中的 Inputs 区域中，首先选中刚才设置的 X 值，然后单击【View】按钮，得到一个新的窗口，如图 2-69 所示。在这个窗口中显示输入向量 X 的值，在此可以对其进行修改。单击【Manager】按钮，便可再次返回到【Network/Data Manager】窗口中。在【Network/Data Manager】窗口中，利用【Delete】按钮，则会将刚才输入的向量 X 从图形用户接口工作空间中删除。

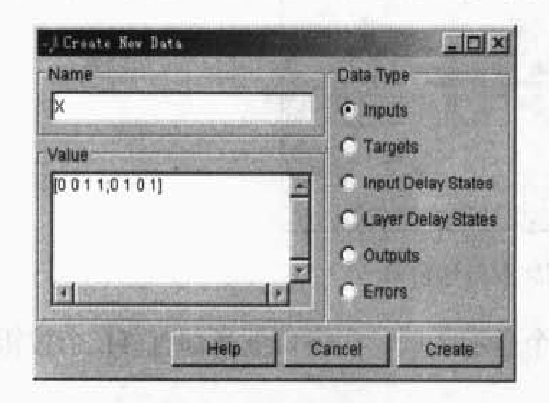


图 2-68 数据输入窗口

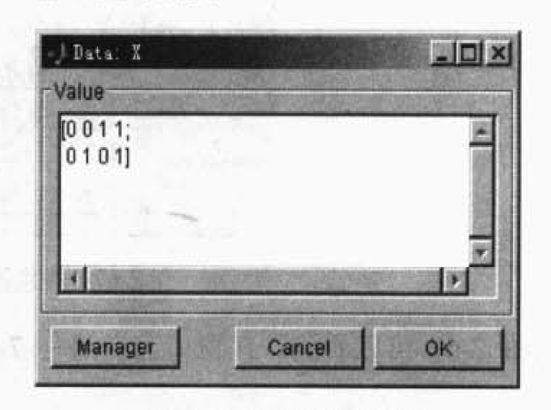


图 2-69 输入数据 X 值

根据同样的步骤设置目标向量 T，但此时要确保如图 2-68 所示的【Create New Data】数据输入窗口右边单选框中的 Targets 选项被选中。

（3）建立网络

在如图 2-67 所示的图形界面中单击【New Network】按钮，弹出一个名为【Create New Network】的网络设计窗口，如图 2-70 所示。在其【Network Name】区域中输入 ORnet 作为网络名。然后，将网络类型设置为感知机，即在其【Network Type】区域中选择 Perceptron。【Input ranges】区域用于设置输入向量的范围，可以直接在其中填写数字，也可单击【Input ranges】右边的下拉箭头，在下拉列表中选择输入向量 X，它自动地从输入数据 X 中得到两输入向量的范围均为[0 1]，即[0 1;0 1]。

神经元个数(Number of neurons)取 1、传输函数(Transfer function)选 HARDLIM 和学习函数（Learning function）选 LEARNP，如图 2-70 所示。在此窗口中单击【View】按钮，可以看到所建网络结构图，如图 2-71 所示。

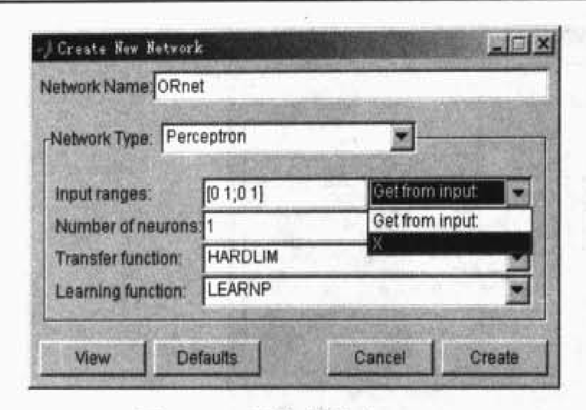


图 2-70 网络设计窗口

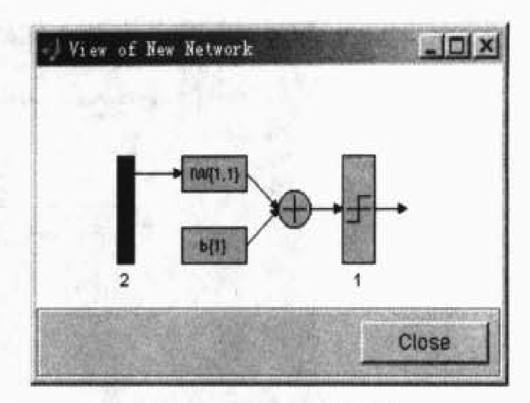


图 2-71 网络结构图

在设置好以上参数后，单击【Create】按钮，就完成了在图形用户接口中建立网络的过程。此时，返回到【Network/Data Manager】窗口中，并在其中的【Networks】区域中将ORnet 作为网络名显示出来。此时，如果用鼠标单击【ORnet】网络名将其选中，则在【Network/Data Manager】窗口中最下面的【Networks only】区域中的4个按钮【Initialize】、【Simulate】、【Train】、【Adapt】被激活，如图 2-72 所示。

利用神经网络编辑器除了可以对感知机（Perceptron）进行设计以外，还可对多种神经网络系统进行设计，如图 2-73 所示。但对于含有隐含层的多层网络，如 BP（Back Propagation）网络的参数设置时要注意，在选择框 Properties for 中的 Layer1 指的是第一个隐含层，而不是输入层，Layer2 指的是第二个隐含层或输出层。因在 MATLAB 神经网络工具箱中的第一层是从隐含层开始的，其后依次为第二、三层。

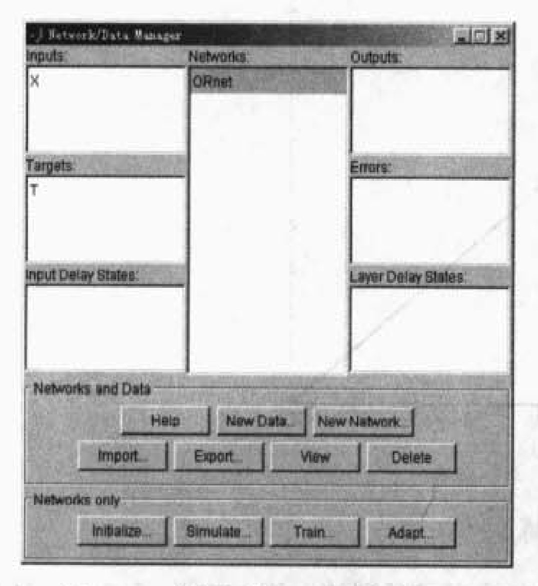


图 2-72 神经网络 ORnet 编辑器

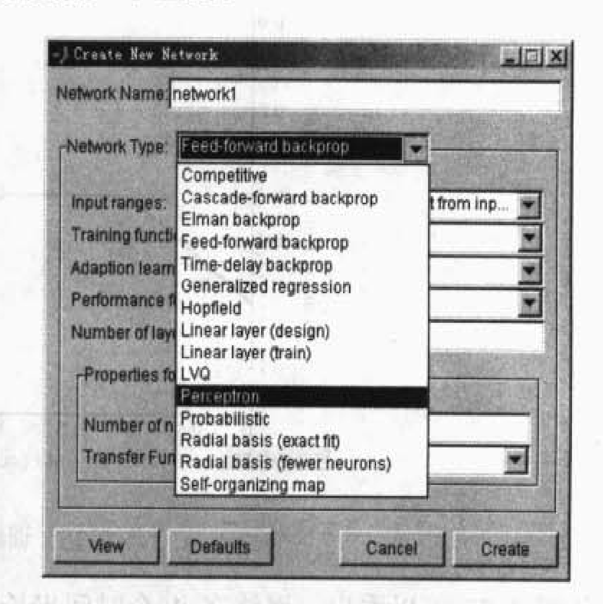


图 2-73 编辑器的 Network Type 窗口

（4）训练网络

在如图 2-72 所示的图形界面中，在确保 ORnet 网络名被选中的情况下，单击【Train】按钮，将弹出一个名为【Network:ORnet】的新窗口，如图 2-74 所示。

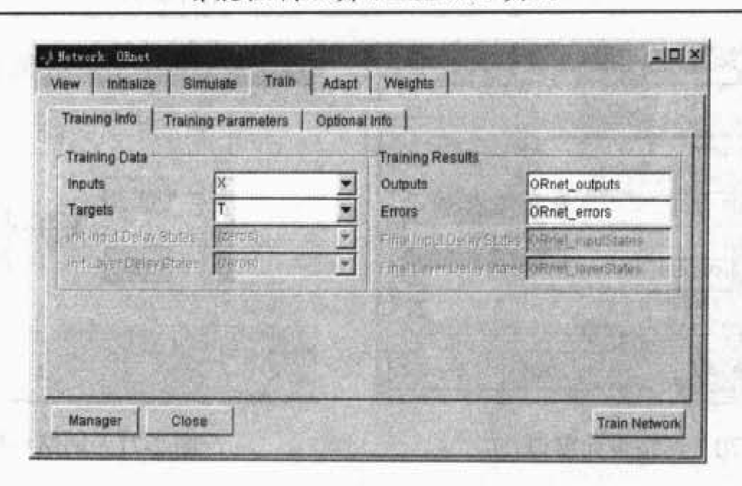


图 2-74　训练窗口

在如图 2-74 所示窗口的【Train】页面的 Training Info 子选项中，将【Inputs】区域选为 X，将【Targets】区域选为 T。【Training Parameters】子选项用于设置训练步数、误差目标等参数；【Optional Info】子选项用于设置一些其他参数。

另外，单击如图 2-74 所示左上角的【View】选项，也可以看到所建网络 ORnet 的结构图。【Initialize】选项可改变或检查网络的初始值；【Weights】选项可设置网络的权值。

当设置好以上参数后，单击如图 2-74 所示窗口右下角的【Train Network】按钮，就开始了网络的训练。训练过程如图 2-75 所示。

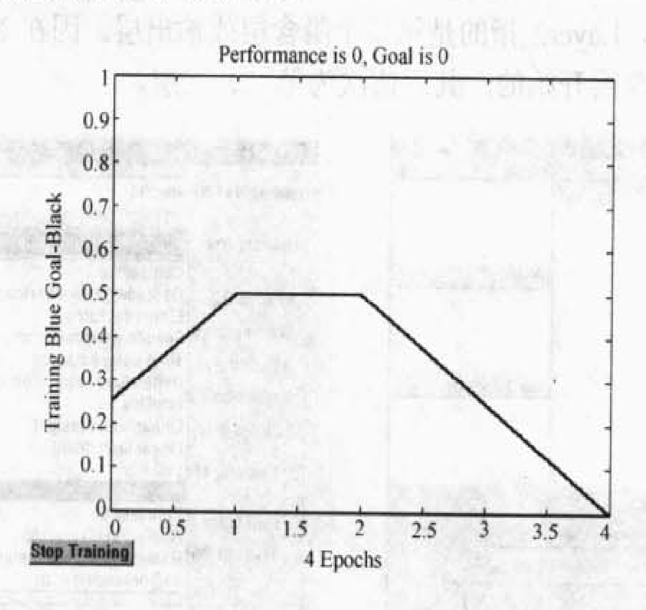


图 2-75　训练过程

从图 2-75 可以看出，网络在 4 个时间步长中就完成了训练，误差变为了 0。这是对感知机网络而言的，对于其他网络来说，通常是不能将误差变为 0 的，甚至误差范围还比较大。在那种情况下，就不能像在这里一样在线性坐标图上绘出训练过程，而应在对数坐标图上绘出。

（5）仿真网络

在如图 2-72 所示的图形界面中，在确保 ORnet 网络名被选中的情况下，单击【Simulate】按钮，或者在如图 2-74 所示的训练窗口中，单击【Simulate】按钮，将弹出一个名为【Network:Ornet】的新窗口，如图 2-76 所示。

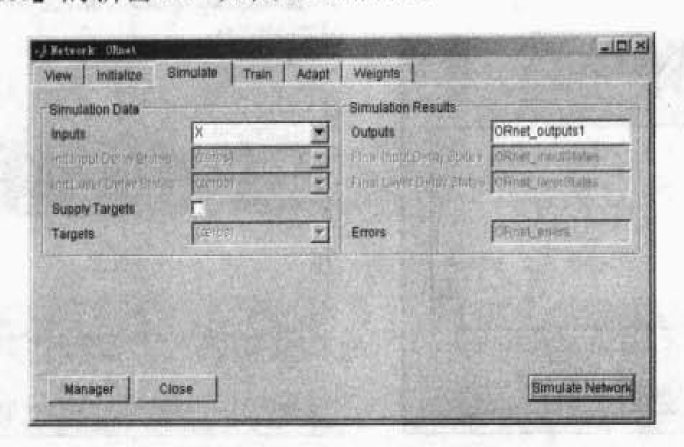


图 2-76　仿真参数设置窗口

在如图 2-76 所示的窗口中，将【Inputs】区域选为 X，将【Outputs】栏中的变量名改为：ORnet_outputs1，以便与训练的输出数据相区分。当设置好以上参数后，单击如图 2-76 所示窗口右下角的【Simulate Network】按钮，就开始了网络的仿真。仿真结束后返回到如图 2-77 所示的【Network/Data Manager】窗口。这时在【Outputs】区域中将增加一个新的变量 ORnet_outputs1，双击这个变量，将会弹出一个【Data:ORnet_outputs1】窗口，如图 2-78 所示。

图 2-77　【Network/Data Manager】窗口

从如图 2-78 所示中可以看出，网络的仿真输出结果等于目标值 T，即网络刚好实现了“或”的功能。

（6）将数据导出到 MATLAB 工作空间

在如图 2-72 所示的窗口中，单击【Export】按钮，将弹出一个名为【Export or Save from Network/Data Manager】的新窗口，如图 2-79 所示。

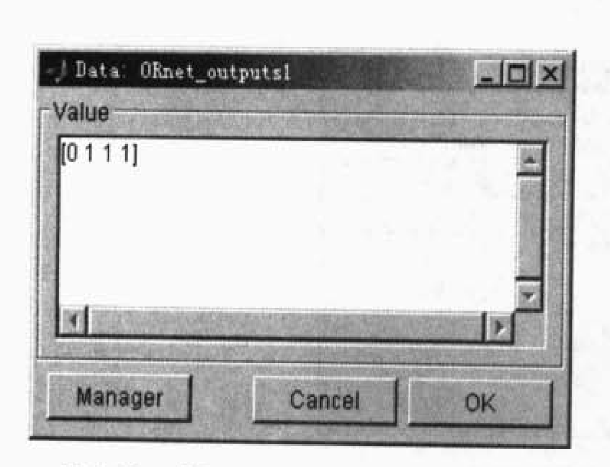


图 2-78 【Data:ORnet_outputs1】窗口

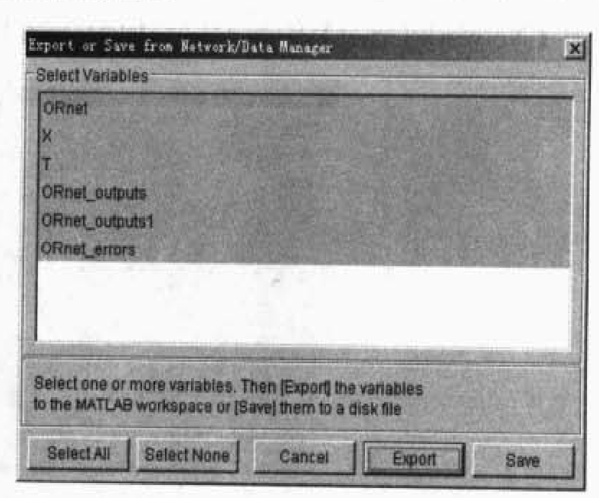


图 2-79 【Export or Save from Network/Data Manager】窗口

在这个窗口中，列出了刚才建立的网络及相关的变量。可以在其中选择需要导出到 MATLAB 工作空间的变量。最后单击【Export】按钮，便可将选中的变量导出到 MATLAB 工作空间。

如果网络 ORnet 已经被导出到 MATLAB 工作空间中，则可利用以下命令得到该网络的权值矩阵和偏值。

```
>>w=ORnet.iw{1,1},b=ORnet.b{1}
```

其结果显示为

```
w=
     1   1
b=
    -1
```

另外，利用如图 2-79 所示窗口的右下角的【Save】按钮，可将选中的变量以 MAT 文件的形式保存到磁盘中。

（7）从 MATLAB 工作空间中导入数据

首先在 MATLAB 命令工作空间中利用以下命令定义一个变量，即

```
>>r=[0;1;1;1];
```

然后在如图 2-72 所示的窗口中单击【Import】按钮，将弹出一个名为【Import or Load to Network/Data Manager】的新窗口。在此窗口中设置【Source】项为：Import from MATLAB workspace；选择【Select a Variable】项中的变量 r，并在【Destination】项中设置目标变量名为：r（注意这两个 r 处于不同的工作空间中，此处的变量名也可设置为其他，

如X)；在【Import As】区域中选择Inputs。如图2-80所示。

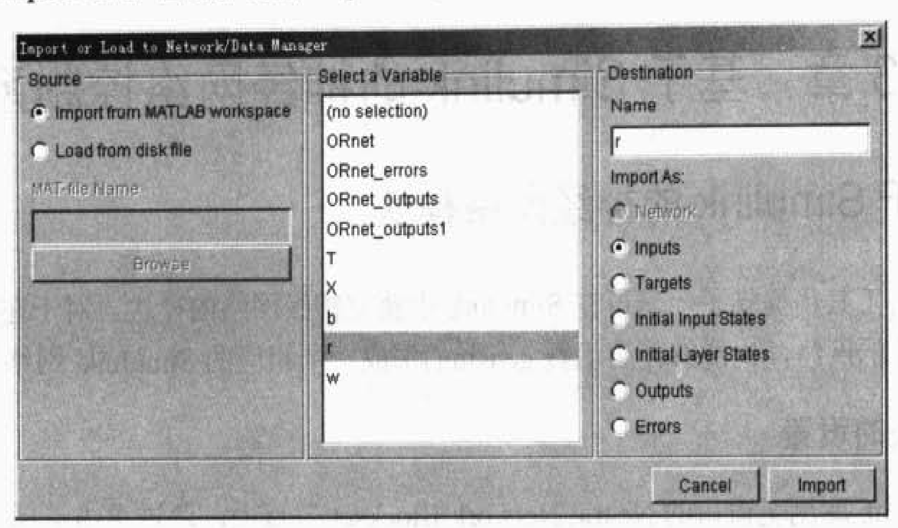


图2-80 【Import or Load to Network/Data Manager】窗口

当设置好以上参数后，单击如图2-80所示窗口右下角的【Import】按钮，就将此数据导入到了图形用户接口工作空间中了。此时，在【Network/Data Manager】窗口的【Inputs】区域中将可看见新导入的数据r。

另外，利用如图2-80所示窗口【Source】项的【Load from disk file】选项，可从磁盘中调用已有的MAT文件到图形用户接口工作空间中。

第 3 章　基于 Simulink 的神经网络控制系统

3.1　基于 Simulink 的神经网络模块

神经网络工具箱提供了一套可在 Simulink 中建立神经网络的模块，对于在 MATLAB 工作空间中建立的网络，也能够使用函数 gensim()生成一个相应的 Simulink 网络模块。

3.1.1　模块的设置

在 Simulink 库浏览窗口的 Neural Network Blockset 节点上，通过单击鼠标右键后，便可打开如图 3-1 所示的 Neural Network Blockset 模块集窗口。

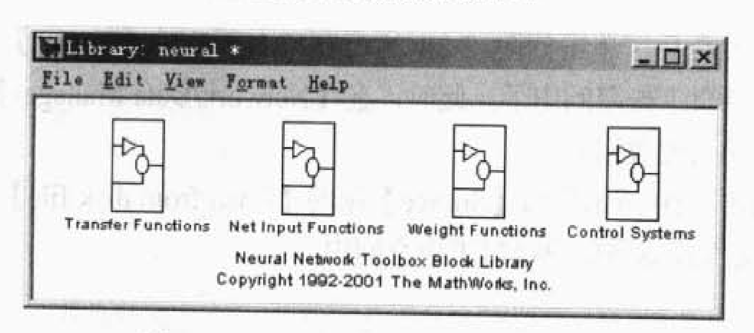


图 3-1　Neural Network Blockset 模块集

Neural Network Blockset 模块集包含了四个模块库，用鼠标的左键双击各个模块库的图标，便可打开相应的模块库。

1．传输函数模块库(Transfer Functions)

用鼠标的左键双击 Transfer Functions 模块库的图标，便可打开如图 3-2 所示的传输函数模块库窗口。传输函数模块库中的任意一个模块都能够接受一个网络输入向量，并且相应地产生一个输出向量，这个输出向量的组数和输入向量相同。

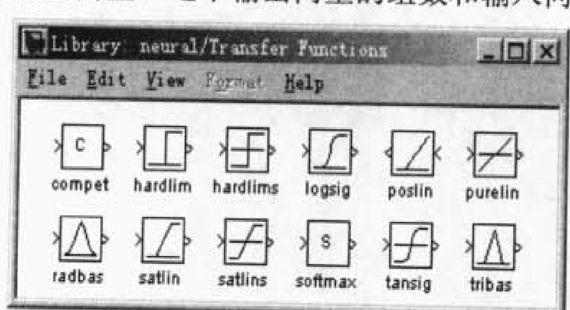


图 3-2　传输函数模块库窗口

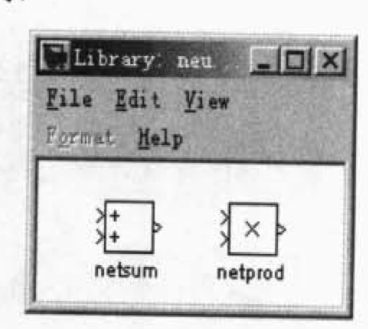


图 3-3　网络输入模块库窗口

2. 网络输入模块库(Net Input Functions)

用鼠标的左键双击 Net Input Functions 模块库的图标，便可打开如图 3-3 所示的网络输入模块库窗口。

网络输入模块库中的每一个模块都能够接受任意数目的加权输入向量、加权的层输出向量及偏值向量，并且返回一个网络输入向量。

3. 权值模块库(Weight Functions)

用鼠标的左键双击 Weight Functions 模块库的图标，便可打开如图 3-4 所示的权值模块库窗口。权值模块库中的每个模块都以一个神经元权值向量作为输入，并将其与一个输入向量（或者是某一层的输出向量）进行运算，得到神经元的加权输入值。

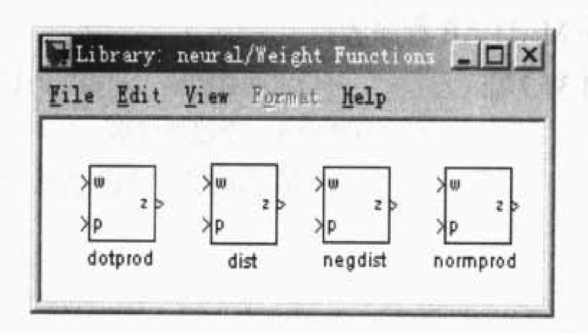


图 3-4　权值模块库窗口

上面的这些模块需要的权值向量必须定义为列向量。原因是 Simulink 中的信号可以为列向量，但是不能为矩阵或者行向量。

4. 控制系统模块库(Control Systems)

用鼠标的左键双击 Control Systems 模块库的图标，便可打开如图 3-5 所示的控制系统模块库窗口。

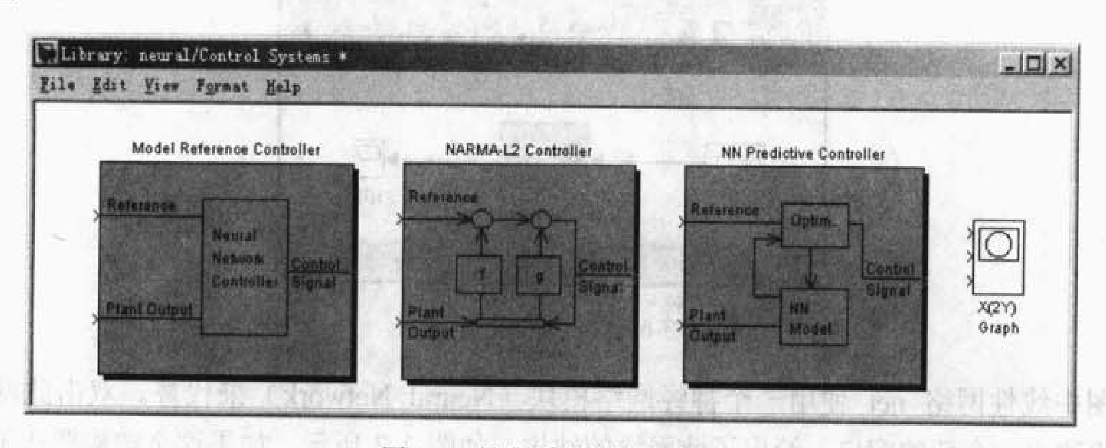


图 3-5　控制系统模块库窗口

神经网络的控制系统模块库包含三个控制器和一个示波器。它们的使用方法将在下一节专门介绍。

3.1.2 模块的生成

在 MATLAB 工作空间中，利用函数 gensim()，能对一个神经网络生成模块化描述，从而可在 Simulink 中对其进行仿真。gensim() 函数的调用格式为

gensim(net,st)

式中，第一个参数指定了 MATLAB 工作空间中需要生成模块化描述的网络，第二个参数指定了采样时间，通常情况下为一正数。如果网络没有与输入权值或者层中权值相关的延迟，则指定第二个参数为–1，那么函数 gensim()将生成一个连续采样的网络。

例 3-1 设计一个线性网络，并生成其模块化描述。定义网络的输入为：X=[1 2 3 4 5]，相应的目标为：T=[1 3 5 7 9]。

解：实现以上任务的 MATLAB 程序为：

```
X=[1 2 3 4 5]; T=[1 3 5 7 9];          %给定网络的输入和相应的目标值
net=newlind(X,T);                       %设计一个线性网络
y=sim(net,X)                            %利用原始的输入数据测试网络
```

其结果显示为

```
y=
    1  3  5  7  9
```

可以看出，网络已经正确地解决了问题。

```
>>gensim(net,–1) %生成网络 net 的模块化描述。
```

调用函数 gensim()后，将生成一个新的窗口。这个窗口包含有一个线性网络，这个线性网络连接了一个采样输入和一个示波器，如图 3-6 所示。

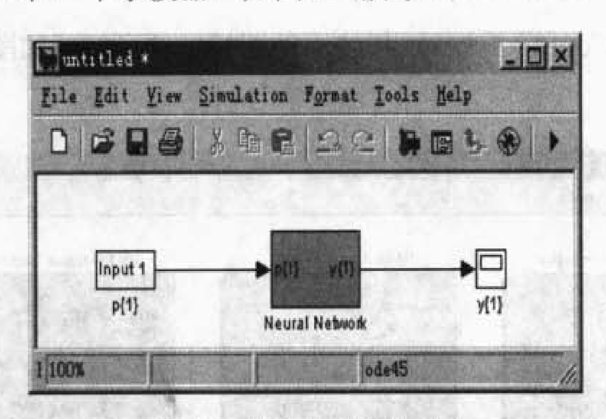


图 3-6　线性网络的仿真图

图中线性网络 net 使用一个神经网络模块（Neural Network）来代替。双击此网络模块，将弹出一个新的窗口，绘出了此网络的结构，如图 3-7 所示。如果这个结构图还不够具体，不能满足要求，还可以进一步在其基础上双击需要了解的部分。此时，将继续弹出新的窗口，在此新窗口中出现了刚才单击部分的更仔细的结构，图 3-8 则为 Layer1 模块的详细结构。这样一直下去，直到出现的窗口为属性设置窗口为止。

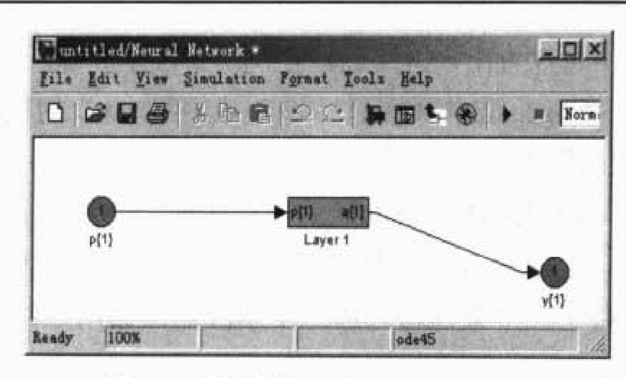


图 3-7 线性神经网络 net 的结构

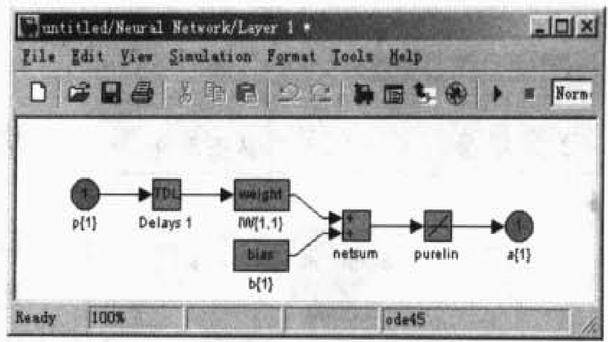


图 3-8 Layer1 模块的详细结构

在如图 3-6 所示的窗口中，单击 p{1}模块，打开其属性设置窗口，如图 3-9 所示。该模块实际上是一个标准的常量模块。在窗口中将 Constant value 区域的值改为 2，然后单击【OK】按钮。再回到如图 3-6 所示的窗口中，启动仿真，便可在示波器中观察到网络的输出信号，如图 3-10 所示。

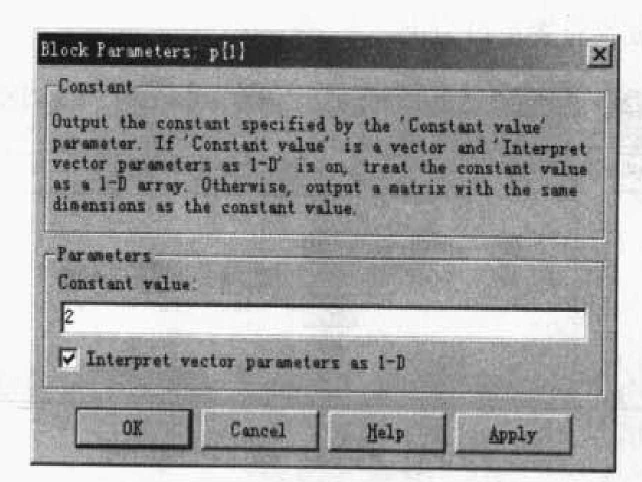


图 3-9 p{1}模块的属性设置窗口

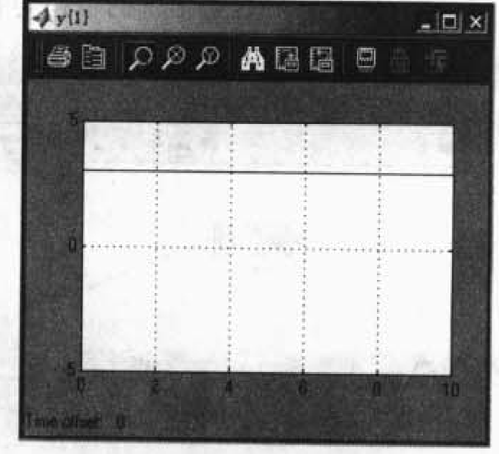


图 3-10 网络的输出信号

从图 3-10 中可看出，网络的输出信号为 3，这与在 MATLAB 工作空间中使用函数 sim()得到的结果是一致的，即输入为 2 时，输出为 3。

例 3-2 建立一个感知机网络，使其能够完成“与”的功能。

解：为了完成“与”函数，建立一个两输入、单输出的感知机网络。设输入向量为：X=[0 0 1 1;0 1 0 1]，相应的目标向量为：T=[0 0 0 1]。实现以上任务的 MATLAB 程序为：

```
X=[0 0 1 1;0 1 0 1]; T=[0 0 0 1];      %给定网络的输入和相应的目标值
net1=newp([-1 1;-1 1],1);             %设计一个感知机
net1=train(net1,X,T);                 %利用输入向量和目标向量训练感知机
y=sim(net1,X)                         %利用原始的输入数据测试网络
```

其结果显示为

```
y=
    0  0  0  1
```

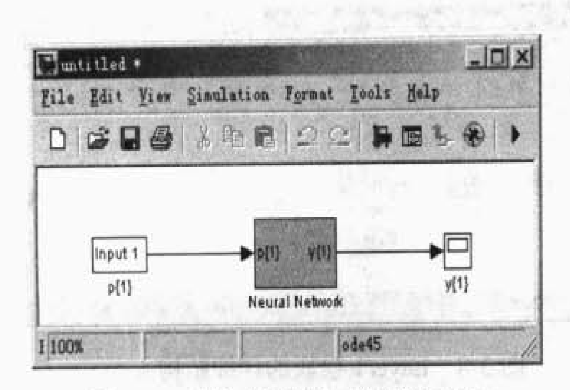


图 3-11　感知机神经网络的仿真图

可以看出，网络已经正确地解决了问题。

>>gensim(net1,−1)　%生成网络 net1 的模块化描述。

调用函数 gensim()后，将生成一个新的窗口。这个窗口包含有一个感知机，这个感知机连接了一个采样输入和一个示波器，如图 3-11 所示。

图中感知机 net1 使用一个神经网络模块（Neural Network）来代替。双击此网络模块，将弹出一个新的窗口，绘出了此网络的结构，如图 3-12 所示。如果这个结构图还不够具体，不能满足要求，还可以进一步在其基础上双击需要了解的部分。此时，将继续弹出新的窗口，在此新窗口中出现了刚才单击部分的更仔细的结构，图 3-13 则为 Layer1 模块的详细结构。这样一直下去，直到出现的窗口为属性设置窗口为止。

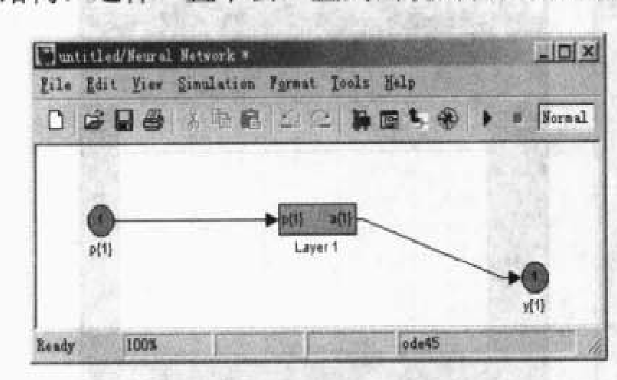


图 3-12　感知机 net1 模块的结构

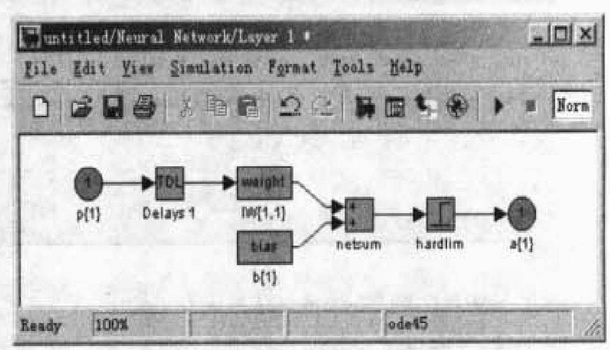


图 3-13　Layer1 模块的详细结构

在如图 3-11 所示的窗口中，单击 p{1}模块，打开其属性设置窗口，如图 3-14 所示。该模块实际上是一个标准的常量模块。在窗口中将 Constant value 区域的值改为[1;0]，然后单击【OK】按钮。再回到如图 3-11 所示的窗口中，启动仿真，便可在示波器中观察到网络的输出信号，如图 3-15 所示。

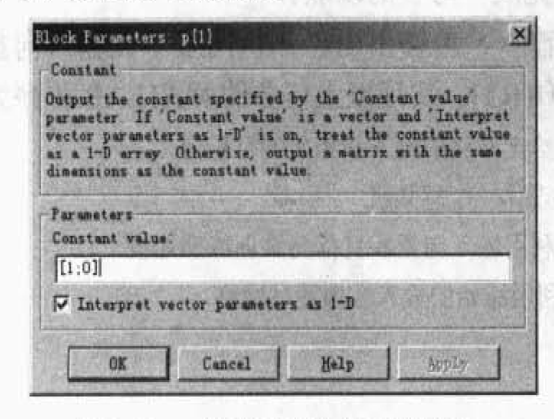


图 3-14　p{1}模块的属性设置窗口

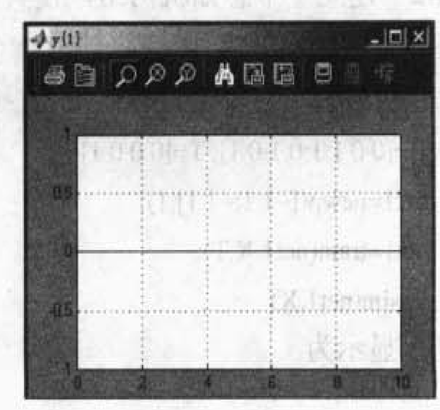


图 3-15　网络的输出信号

从图 3-15 中可看出，网络的输出信号为 0，这与在 MATLAB 工作空间中使用函数 sim() 得到的结果是一致的，即输入为[1;0]时，输出为 0。

3.2 基于 Simulink 的三种典型神经网络控制系统

神经网络在系统辨识和动态系统控制中已经得到了非常成功的使用。由于神经网络具有全局逼近能力，使得其在对非线性系统建模和对一般情况下的非线性控制器的实现等方面应用的比较普遍。本节将介绍三种在神经网络工具箱的控制系统模块（Control Systems）中，利用 Simulink 实现的比较典型的神经网络结构，它们常用于预测和控制，并已在 MATLAB 对应的神经网络工具箱中给出了相应的实现方法。这三种神经网络结构分别是：

- 神经网络模型预测控制（NN Predictive Controller）；
- 反馈线性化控制（NARMA-L2 Controller）；
- 模型参考控制（Model Reference Controller）。

使用神经网络进行控制时，通常有两个步骤：系统辨识和控制设计。

在系统辨识阶段，主要任务是对需要控制的系统建立神经网络模型；在控制设计阶段，主要使用神经网络模型来设计（训练）控制器。在本节将要介绍的三种控制网络结构中，系统辨识阶段是相同的，而控制设计阶段则各不相同。

对于模型预测控制，系统模型用于预测系统未来的行为，并且找到最优的算法，用于选择控制输入，以优化未来的性能。

对于 NARMA-L2（反馈线性化）控制，控制器仅仅是将系统模型进行重整。

对于模型参考控制，控制器是一个神经网络，它被训练并用于控制系统，使得系统跟踪一个参考模型，这个神经网络系统模型在控制器训练中起辅助作用。

3.2.1 神经网络模型预测控制

1．模型预测控制理论

神经网络预测控制器是使用非线性神经网络模型来预测未来模型性能。控制器计算控制输入，而控制输入在未来一段指定的时间内将最优化模型性能。模型预测第一步是要建立神经网络模型（系统辨识）；第二步是使用控制器来预测未来神经网络性能。

1）系统辨识

模型预测的第一步就是训练神经网络未来表示网络的动态机制。模型输出与神经网络输出之间的预测误差，用来作为神经网络的训练信号。该过程如图 3-16 所示。

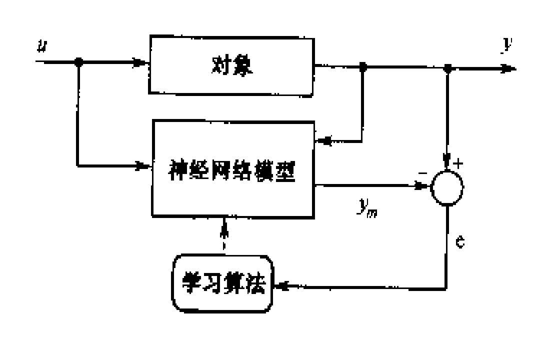


图 3-16　训练神经网络

神经网络模型利用当前输入和当前输出预测神经网络未来输出值。神经网络模型结构如图 3-17 所示。该网络可以采用批量在线训练。

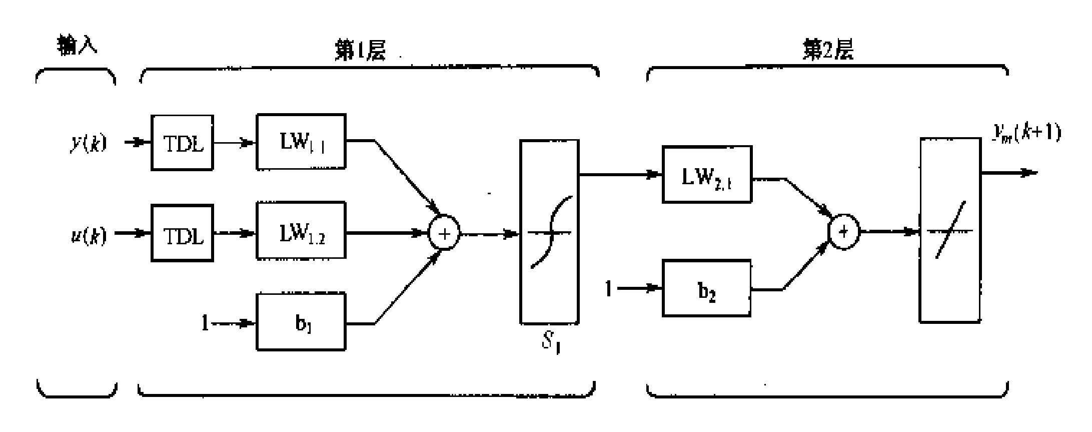


图 3-17　神经网络模型结构

2）模型预测

模型预测方法是基于水平后退的方法，神经网络模型预测在指定时间内预测模型响应。预测使用数字最优化程序来确定控制信号，通过最优化如下的性能准则函数，即

$$J=\sum_{j=1}^{N_2}[y_r(k+j)-y_m(k+j)]^2+\rho\sum_{j=1}^{N_u}[u(k+j-1)-u(k+j-2)]^2$$

式中，N_2 为预测时域长度；N_u 为控制时域长度；$u(t)$为控制信号；y_r 为期望响应；y_m 为网络模型响应；ρ 为控制量加权系数。

图 3-18 为模型预测控制的过程。控制器由神经网络模型和最优化方块组成，最优化方块确定 u（通过最小化 J），最优 u 值作为神经网络模型的输入，控制器方块可用 Simulink 实现。

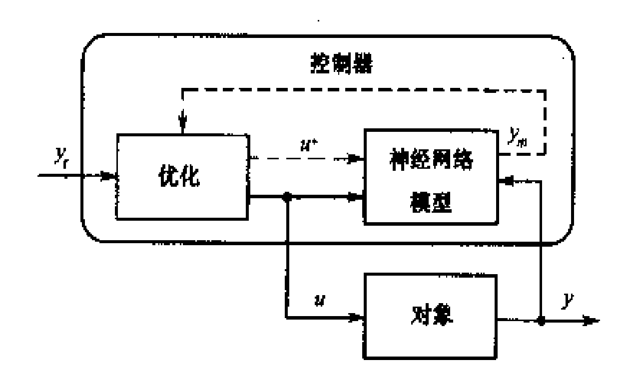


图 3-18　模型预测控制的过程

2. 模型预测神经网络控制实例分析——搅拌器控制系统

在 MATLAB 神经网络工具箱中实现的神经网络预测控制器，使用了一个非线性系统模型，用于预测系统未来的性能。接下来这个控制器将计算控制输入，用于在某个未来的时间区间里优化系统的性能。进行模型预测控制首先要建立系统的模型，然后使用控制器来预测

未来的性能。下面将结合 MATLAB 神经网络工具箱中提供的一个演示实例，介绍 Simulink 中的实现过程。

1）问题的描述

要讨论的问题基于一个搅拌器（CSTR），如图 3-19 所示。

对于这个系统，其动力学模型为

$$\frac{\mathrm{d}h(t)}{\mathrm{d}t}=w_1(t)+w_2(t)-0.2\sqrt{h(t)}$$

$$\frac{\mathrm{d}C_b(t)}{\mathrm{d}t}=(C_{b1}-C_b(t))\frac{w_1(t)}{h(t)}+(C_{b2}-C_b(t))\frac{w_2(t)}{h(t)}-\frac{k_1C_b(t)}{(1+k_2C_b(t))^2}$$

式中，$h(t)$为液面高度；$C_b(t)$为产品输出浓度；$w_1(t)$为浓缩液 C_{b1} 的输入流速；$w_2(t)$为稀释液 C_{b2} 的输入流速。输入浓度设定为 C_{b1}=24.9，C_{b2}=0.1。消耗常量设置为 k_1=1，k_2=1。

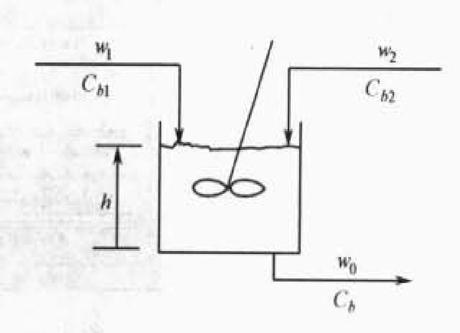


图 3-19 搅拌器

控制的目标通过调节流速 $w_2(t)$来保持产品浓度。为了简化演示过程，不妨设 $w_1(t)$=0.1。在本例中不考虑液面高度 $h(t)$。

2）建立模型

MATLAB 神经网络工具箱提供了这个演示实例。只需在 MATLAB 命令窗口中输入：predcstr，就会自动地调用 Simulink，并且产生如图 3-20 所示的模型窗口。其中，神经网络预测控制模块（NN Predictive Controller）和 X(2Y) Graph 模块由神经网络模块集（Neural Network Blockset）中的控制系统模块库（Control Systems）复制而来。

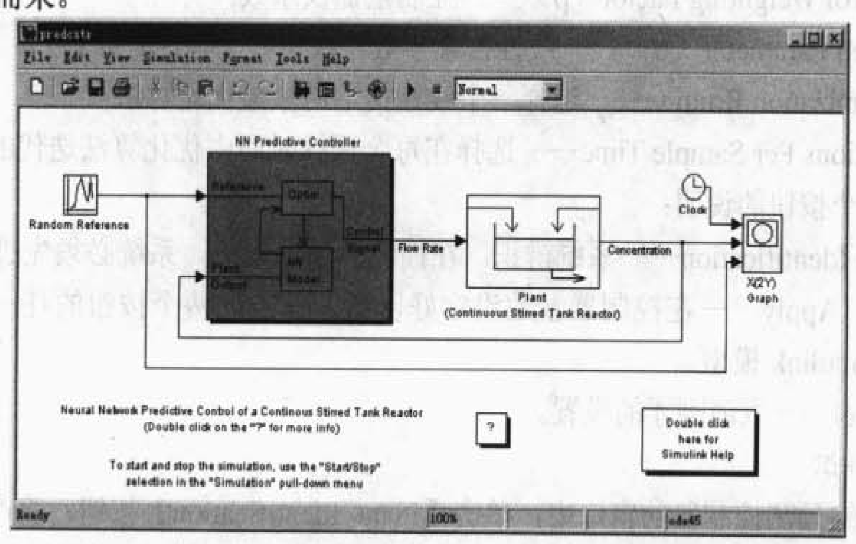


图 3-20 模型窗口

图 3-20 中的 Plant（Continuous Stirred Tank Reactor）模块包含了搅拌器系统的 Simulink

模型。双击这个模块，即可得到具体的 Simulink 实现。此处将不再深入讨论。

NN Predictive Controller 模块的 Control Signal 端连接到搅拌器系统模型的输入端，同时搅拌器系统模型的输出端连接到 NN Predictive Controller 模块的 Plant Output 端，参考信号连接到 NN Predictive Controller 模块的 Reference 端。

双击 NN Predctive Controller 模块，将会产生一个神经网络预测控制器参数设置窗口（Neural Network Predctive Control），如图 3-21 所示。这个窗口用于设计模型预测控制器。

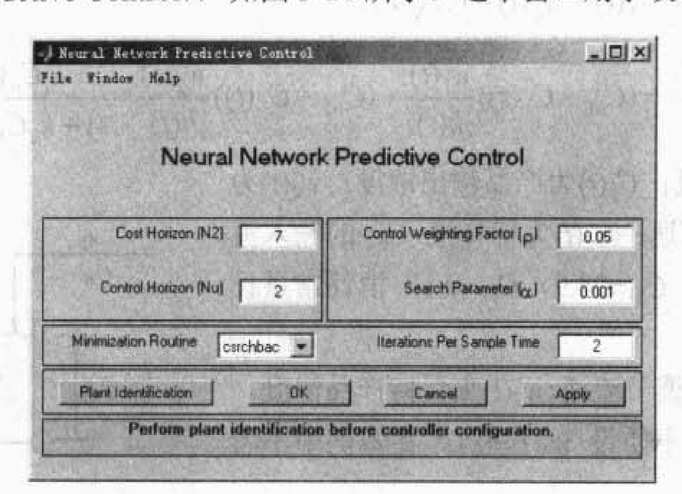


图 3-21　神经网络预测控制器参数设置窗口

在这个窗口中，有多项参数可以调整，用于改变预测控制算法中的有关参数。将鼠标移到相应的位置，就会出现对这一参数的说明。现将这些说明分别加以解释。

（1）Cost Honizon（N_2）——预测时域长度；

（2）Control Honizon（N_u）——控制时域长度；

（3）Control Weighting Factor（ρ）——控制量加权系数；

（4）Search Parameter（α）——线性搜索参数，决定搜索何时停止；

（5）Minimization Routine——选择一个线性搜索用做最优化算法；

（6）Iterations Per Sample Time——选择在每个采样时间中优化算法迭代的次数。

下面是四个按钮的说明：

（1）Plant Identification——系统辨识。在控制器使用之前，系统必须先进行辨识。

（2）OK、Apply——在控制器参数设定好以后，单击这两个按钮的任一个都可以将这些参数导入 Simulink 模型。

（3）Cancel——取消刚才的设置。

3）系统辨识

在神经网络预测控制器的窗口中，单击【Plant Identification】按钮，将产生一个模型辨识参数设置窗口（Plant Identification），用于设置系统辨识的参数，如图 3-22 所示。

在控制器使用以前，必须首先利用辨识技术建立神经网络模型。这个模型预测系统未

来的输出值。优化算法使用这些预测值来决定控制输入，以优化未来的性能。系统的神经网络模型有一个隐含层。这个隐含层的大小、输入和输出的时延及训练函数，都在如图 3-22 所示的窗口中设置。可以选择 BP 网络中的任意训练函数来训练网络模型。

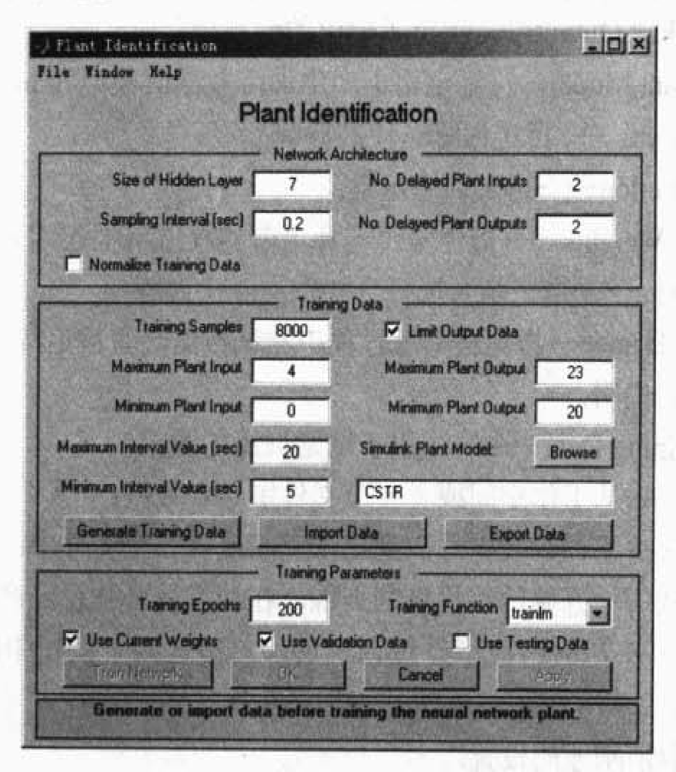


图 3-22 模型辨识参数设置窗口

在窗口菜单中有一项 File，其包含的子项中有两项用于导入和导出系统模型对应的网络。

与如图 3-21 所示类似，在如图 3-22 所示的窗口中，有很多参数需要设置。将鼠标移到相应的位置，也会出现对这些参数的说明，现将这些参数分别加以解释。

（1）Size of Hidden Layer——设置在系统模型网络隐含层中的神经元数；

（2）Sampling Interval（sec）——指定程序从 Simulink 模型中采集数据的间隔；

（3）No. Delayed Plant Inputs——指定了加到系统网络模型的输入延迟；

（4）No. Delayed Plant Outputs——指定了加到系统网络模型的输出延迟；

（5）Normalize Training Data——指定是否使用 premnmx 函数来将数据标准化；

（6）Training Samples——指定了为训练而产生的数据点的数目；

（7）Maximum Plant Input——指定了随机输入的最大值；

（8）Minimum Plant Input——指定了随机输入的最小值；

（9）Maximum Interval Value（sec）——指定一个最大的间隔，在这个间隔中，随机输入将保持不变；

（10）Minimum Interval Value（sec）——指定一个最小的间隔，在这个间隔中，随机输

入将保持不变；

（11）Limit Output Data——用于选择系统输出是否为有界值；

（12）Maximum Plant Output——指定了输出的最大值；

（13）Minimum Plant Output——指定了输出的最小值；

（14）Simulink Plant Model——指定用于产生训练数据的模型（.mdl 文件）；

（15）Training Epochs——指定训练迭代的次数；

（16）Training Function——指定训练函数；

（17）Use Current Weights——指定是否选择当前的权值用于连续训练；

（18）Use Valid Data——指定是否选择合法数据停止训练；

（19）Use Testing Data——指定在训练过程中测试数据是否被追踪。

下面是关于按钮的说明：

（1）Generate Training Data——产生用于网络训练的数据；

（2）Import Data——从工作空间或者一个文件中导入数据；

（3）Export Data——将训练数据导出到工作空间或者一个文件中；

（4）Train Network——开始网络模型的训练，在训练前必须已经产生或者导入了数据；

（5）OK、Apply——在网络模型经过训练后，单击这两个按钮中的任一个都可以将网络导入 Simulink 模型；

（6）Cancel——取消刚才的设置。

系统辨识分为两步：第一步为产生训练数据，第二步为训练网络模型。在模型辨识窗口中，设置好相应的参数后，单击【Generate Training Data】按钮，程序就会通过对 Simulink 网络模型提供一系列随机阶跃信号，来产生训练数据。图 3-23 显示了这些训练数据。

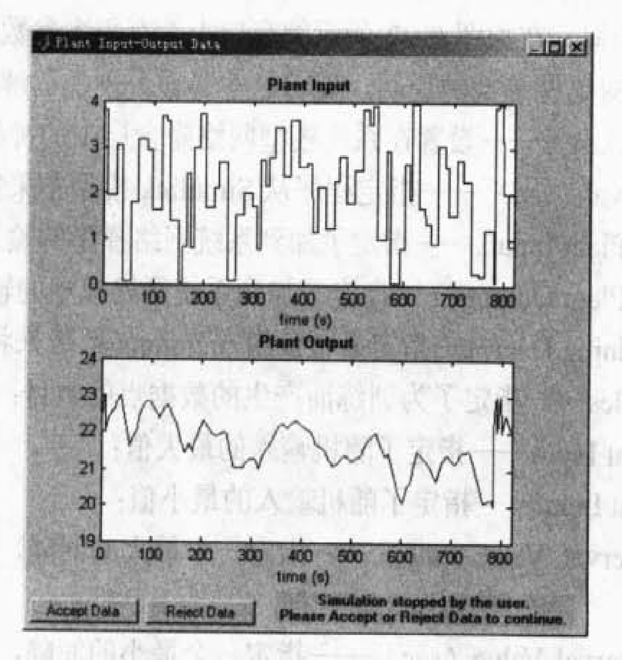


图 3-23 训练数据

在图 3-23 中，有两个按钮，一个为【Accept data】按钮，单击这个按钮，就接受了这些训练数据。另一个为【Refuse Data】按钮，如果单击这个按钮，将会放弃这些训练数据返回到系统辨识窗口，并且可以重新开始。

在图 3-23 的训练数据窗口中，单击【Accept Data】按钮，然后再在模型辨识窗口图 3-22 中单击【Train Network】按钮，网络模型开始训练。训练与选择的训练算法有关（在此处使用的是 trainlm）。

在训练结束后，相应的结果被显示出来，如图 3-24 及图 3-25 所示。

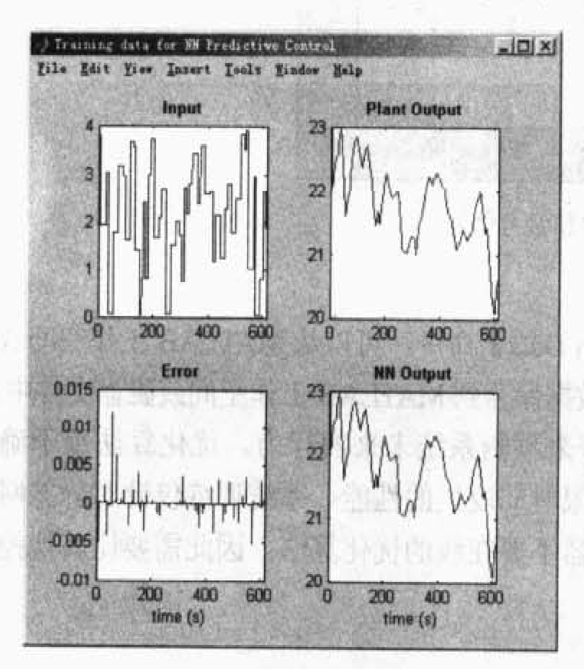


图 3-24　训练数据

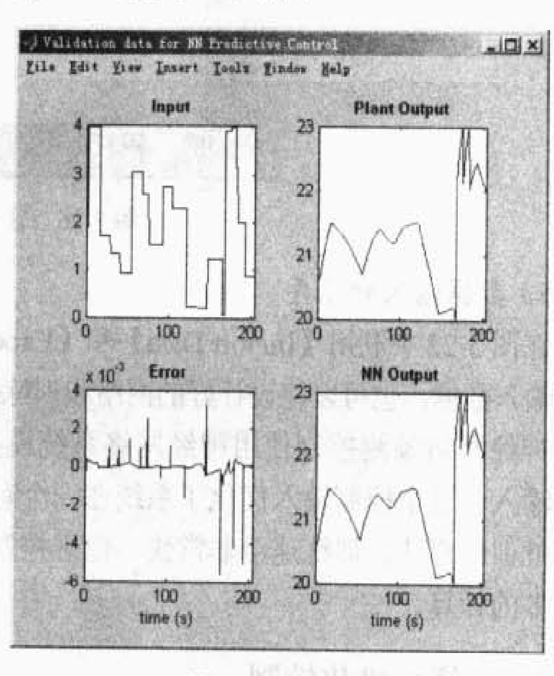


图 3-25　合法数据

图 3-24 显示的是训练数据，图 3-25 显示的是合法数据。在每个图中，左上角的图显示了随机输入信号的阶跃高度和宽度；右上角的图显示了被控对象的输出；左下角的图显示了误差，即系统输出与网络模型输出的差别；右下角的图显示了神经网络模型输出。

在网络模型训练后，可以在图 3-22 中单击【Train Network】按钮，继续再次使用同样的数据进行训练，也可以单击【Erase Generated Data】按钮，产生新的数据。在系统辨识窗口图 3-22 中单击【OK】按钮，便可返回到神经网络预测控制窗口图 3-21 中。

若要接收当前的模型，则在神经网络预测控制窗口图 3-21 中单击【OK】按钮，将训练好的神经网络模型导入到 Simulink 模型窗口图 3-20 中的 NN　Predctive Controller 模块，准备对搅拌器模型预测神经网络控制系统进行仿真。

4）系统仿真

在 Simulink 模型窗口图 3-20 中，首先选择【Simulation】菜单中的【Parameter】命令，设置相应的仿真参数，然后从【Simulation】菜单中单击【Start】命令，开始仿真。仿真的过程需要一段时间。当仿真结束时，将会显示出系统的输出和参考信号，如图 3-26 所示。

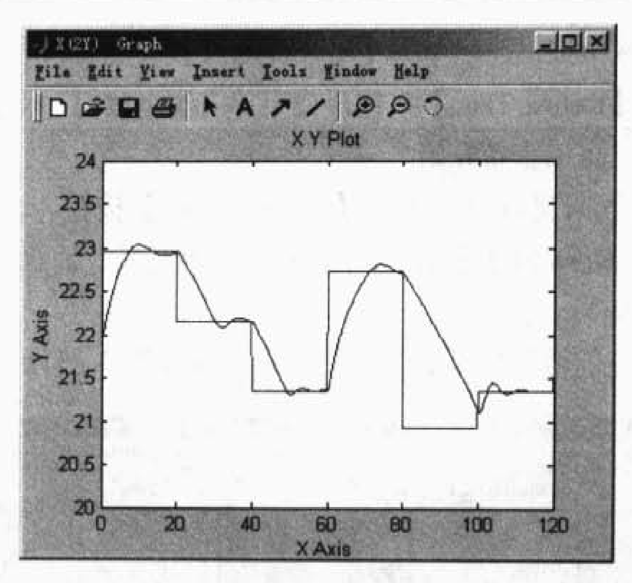


图 3-26　输出和参考信号

5）数据输入和保存

在图 3-22 中利用【Import Data】和【Export Data】命令，可以从 MATLAB 工作空间或磁盘中输入数据，也可以将设计好的网络和训练数据保存到 MATLAB 工作空间或磁盘文件中。

神经网络预测控制使用神经网络系统模型来预测系统未来的行为。优化算法用于确定控制输入，这个控制输入优化了系统在一个有限时间段里的性能。系统训练仅针对静态网络的成批训练算法，训练速度非常快。由于控制器不要在线的优化算法，因此需要比其他控制器更多的计算。

3.2.2　反馈线性化控制

1. 反馈线性化控制理论

反馈线性化（NARMA-L2）的中心思想是通过去掉非线性，将一个非线性系统变换成线性系统。

1）辨识 NARMA-L2 模型

与模型预测控制一样，反馈线性化控制的第一步就是辨识被控制的系统。通过训练一个神经网络来表示系统的前向动态机制，在第一步中首先选择一个模型结构以供使用。一个用来代表一般的离散非线性系统的标准模型是：非线性自回归移动平均模型（NARMA）。它可用下式来表示，即

$$y(k+d)=N[y(k),y(k-1),\cdots,y(k-n+1),u(k),u(k-1),\cdots,u(k-n+1)]$$

式中，$u(k)$表示系统的输入；$y(k)$表示系统的输出。在辨识阶段，训练神经网络使其近似等于非线性函数 N。

如果希望系统输出跟踪一些参考曲线 $y(k+d)=y_r(k+d)$，下一步就是建立一个有如下形式的非线性控制器，即

$$u(k)=G[y(k),y(k-1),\cdots,y(k-n+1),y_r(k+d),u(k-1),\cdots,u(k-n+1)]$$

使用该类控制器的问题是，若想训练一个神经网络用来产生函数 G（最小化均方差），则必须使用动态反馈，且该过程相当慢。由 Narendra 和 Mukhopadhyay 提出的一个解决办法是,使用近似模型来代表系统。在这里使用的控制器模型是基于 NARMA-L2 的近似模型

$$\hat{y}(k+d)=f[y(k),y(k-1),\cdots,y(k-n+1),u(k-1),\cdots,u(k-n+1)]$$
$$+g[y(k),y(k-1),\cdots,y(k-n+1),u(k-1),\cdots,u(k-n+1)]\cdot u(k)$$

该模型是并联形式，控制器输入 $u(k)$没有包含在非线性系统里。这种形式的优点是能解决控制器输入，使系统输出跟踪参考曲线 $y(k+d)=y_r(k+d)$。最终的控制器形式为

$$u(k)=\frac{y_r(k+d)-f[y(k),y(k-1),\cdots,y(k-n+1),u(k-1),\cdots,u(k-n+1)]}{g[y(k),y(k-1),\cdots,y(k-n+1),u(k-1),\cdots,u(k-n+1)]}$$

直接使用该等式会引起实现问题，因为基于输出 $y(k)$的同时必须同时得到 $u(k)$，所以采用下列模型

$$y(k+d)=f[y(k),y(k-1),\cdots,y(k-n+1),u(k),\cdots,u(k-n+1)]$$
$$+g[y(k),y(k-1),\cdots,y(k-n+1),u(k),\cdots,u(k-n+1)]\cdot u(k+1)$$

式中，$d\geqslant 2$。

2）NARMA-L2 控制器

利用 NARMA-L2 模型，可得到如下的控制器，即

$$u(k+1)=\frac{y_r(k+d)-f[y(k),y(k-1),\cdots,y(k-n+1),u(k),u(k-1),\cdots,u(k-n+1)]}{g[y(k),y(k-1),\cdots,y(k-n+1),u(k),u(k-1),\cdots,u(k-n+1)]}$$

式中，$d\geqslant 2$。

2. NARMA-L2（反馈线性化）控制实例分析——磁悬浮控制系统

1）问题的描述

如图 3-27 所示，有一块磁铁，被约束在垂直方向上运动。在其下方有一块电磁铁，通电以后，电磁铁就会对其上的磁铁产生小电磁力作用。目标就是通过控制电磁铁，使得其上的磁铁保持悬浮在空中，不会掉下来。

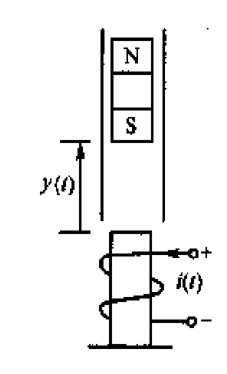


图 3-27 悬浮磁铁控制系统

建立这个实际问题的动力学方程为

$$\frac{d^2y(t)}{dt^2}=-g+\frac{\alpha i^2(t)}{My(t)}-\frac{\beta}{M}\frac{dy(t)}{dt}$$

式中，$y(t)$表示磁铁离电磁铁的距离；$i(t)$代表电磁铁中的电流；M 代表磁铁的质量；g 代表重力加速度；β 代表黏性摩擦系数，它由磁铁所在的容器的材料决定；α 代表场强常数，它由电磁铁上所绕的线圈圈数及磁铁的强度所决定。

2）建立模型

MATLAB 的神经网络工具箱中提供了这个演示实例。只需在 MATLAB 命令窗口中输入

“narmamaglev”，就会自动地调用 Simulink，并且产生如图 3-28 所示的模型窗口。NARMA-L2 控制模块已经被放置在这个模型中。Plant（Magnet Levitation）模块包含了磁悬浮系统的 Simulink 模型。

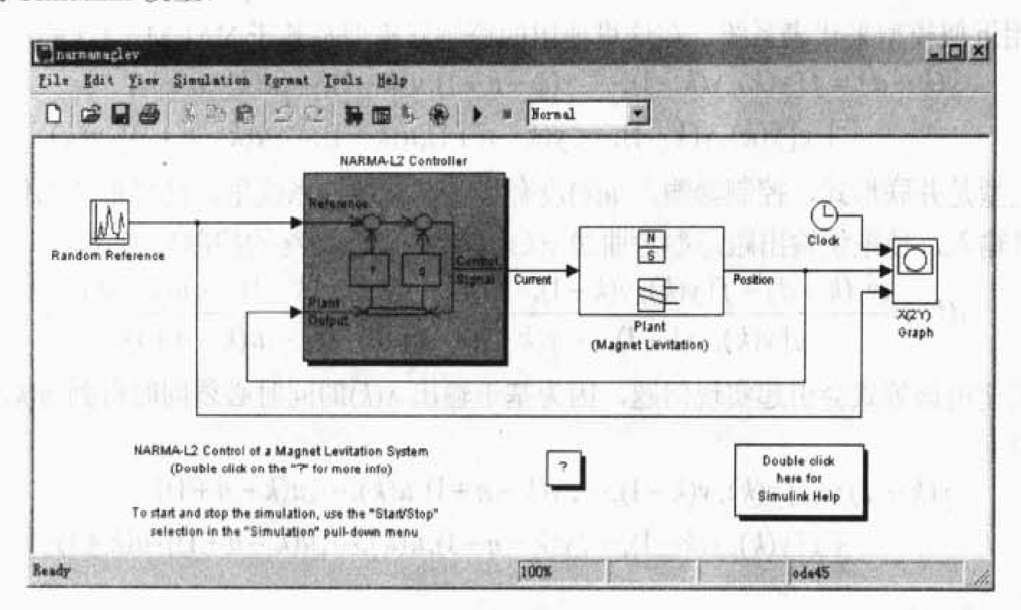


图 3-28　模型窗口

在这个窗口中有一个 NARMA-L2 Controller 模块。这个模块是在神经网络工具箱中生成并复制过来的。这个模块的 Control Signal 端连接到悬浮系统模型的 Current 输入端，此系统模型的 Position 输出端连接到 NARMA-L2 Controller 模块的 Plant Output 端，参考信号连接到该模块的 Reference 端。

3）系统辨识

双击 NARMA-L2 Controller 模块，将会产生一个新的窗口，如图 3-29 所示。这个窗口用于训练 NARMA-L2 模型。这里没有单独的控制器窗口，原因是控制器是直接由模型得到的，在这一点上与模型预测控制不同。

与模型预测控制类似，在使用神经网络控制之前，必须先对系统进行辨识。在如图 3-29 所示的系统辨识参数设置窗口中有很多参数需要设置。系统辨识与前面介绍过的一样分为两步：第一步为产生训练数据；第二步为训练网络模型。操作过程同前，在此不再赘述。

4）系统仿真

系统辨识过程结束后，在 Simulink 模型窗口图 3-28 中，首先选择【Simulation】菜单中的【Parameter】命令，设置相应的仿真参数，然后开始仿真。仿真的过程需要一段时间。当仿真结束时，将会显示出系统的输出和参考信号，如图 3-30 所示。

对于 NARMA-L2（反馈线性化）控制，系统的近似模型转换成标准型计算出的下一个控制输入被用于迫使系统输出跟踪参考信号。由于这种神经网络系统模型使用静态反传算法，因此速度很快。控制器是对系统模型的重整，它需要最小的在线计算。

图 3-29 系统辨识参数设置窗口

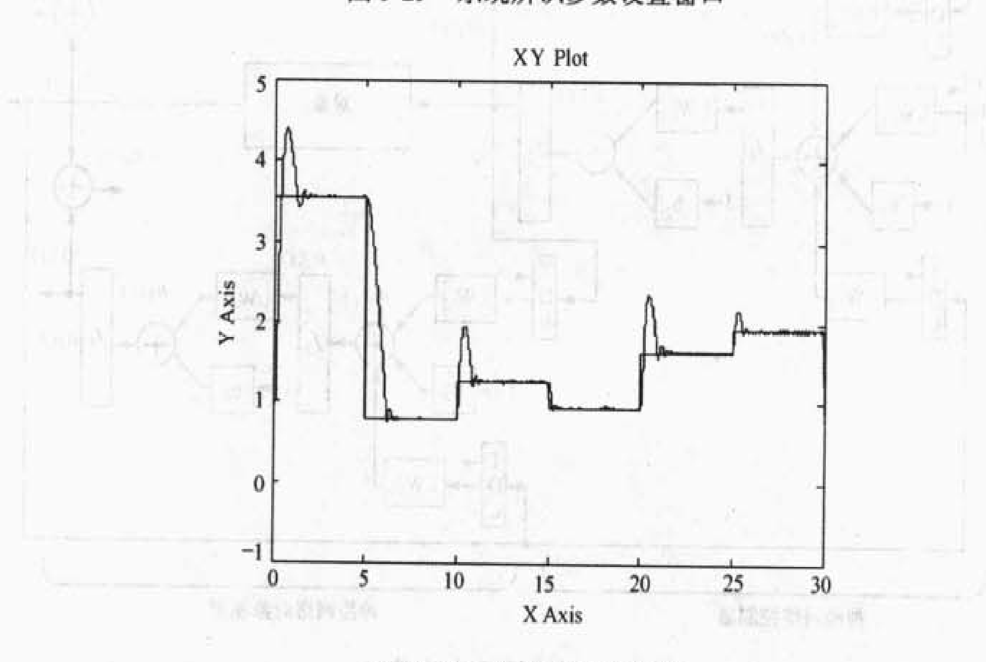


图 3-30 输出和参考信号

3.2.3 模型参考控制

1. 模型参考控制理论

神经模型参考控制采用两个神经网络，即一个控制器网络和一个对象模型网络，如图 3-31 中所示。首先辨识出实验模型，然后训练控制器，使得实验输出跟随参考模型输出。

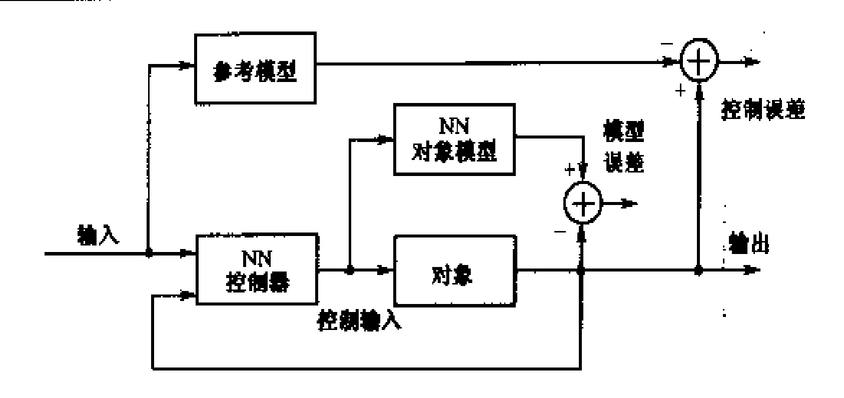


图 3-31 神经模型参考控制系统

2. 模型参考神经网络控制实例分析——机械臂控制系统

图 3-32 为神经网络对象模型，每个网络由两层组成，并且可以选择隐含层的神经元数目。有三组控制器输入：延迟的参考输入、延迟的控制输入和延迟的系统输入。对于每种输入，均可以选择延迟值。通常，随着系统阶次的增加，延迟的数目也增加。对于神经网络系统模型，有两组输入：延迟的控制器输入和延迟的系统输入。

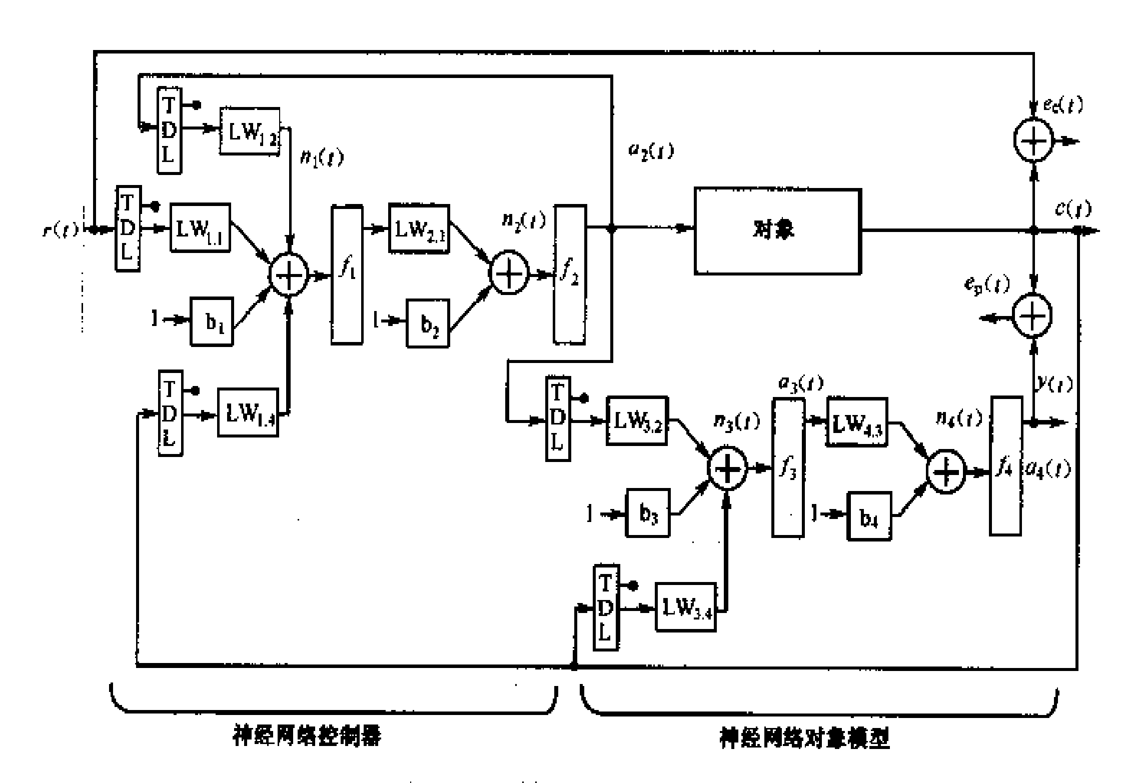


图 3-32 神经网络实验模型

下面结合 MATLAB 神经网络工具箱中提供的一个实例，来介绍神经网络控制器的训练过程。

1）问题的描述

图 3-33 为一个简单的单连接机械臂，目的是控制它的运动。

首先，建立它的运动方程式

$$\frac{\mathrm{d}^2\Phi}{\mathrm{d}t^2}=-10\sin\Phi-2\frac{\mathrm{d}\Phi}{\mathrm{d}t}+u$$

式中，Φ 代表机械臂的角度；u 代表 DC（直流）电机的转矩。目标是训练控制器，使得机械臂能够跟踪参考模型

$$\frac{\mathrm{d}^2y_r}{\mathrm{d}t^2}=-9y_r-6\frac{\mathrm{d}y_r}{\mathrm{d}t}+9r$$

式中，y_r 代表参考模型的输出；r 代表参考信号。

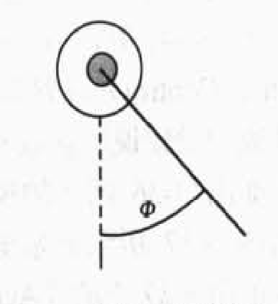


图 3-33　简单的单连接机械臂

2）模型的建立

MATLAB 的神经网络工具箱中提供了这个演示实例。控制器的输入包含了两个延迟参考输入、两个延迟系统输出和一个延迟控制器输出，采样间隔为 0.05s。

只需在 MATLAB 命令行窗口中输入：mrefrobotarm，就会自动地调用 Simulink，并且产生如图 3-34 所示的模型窗口。模型参考控制模块（Model Reference Controller）和机械臂系统的模块已被放置在这个模型中。模型参考控制模块是在神经网络工具箱复制过来的。这个模块的 Control Signal 端连接到机械臂系统模块的 Torque 输入端，系统模型的 Angle 输出端连接到模块的 Plant Output 端，参考信号连接到模块的 Reference 端。机械臂系统模型窗口如图 3-35 所示。

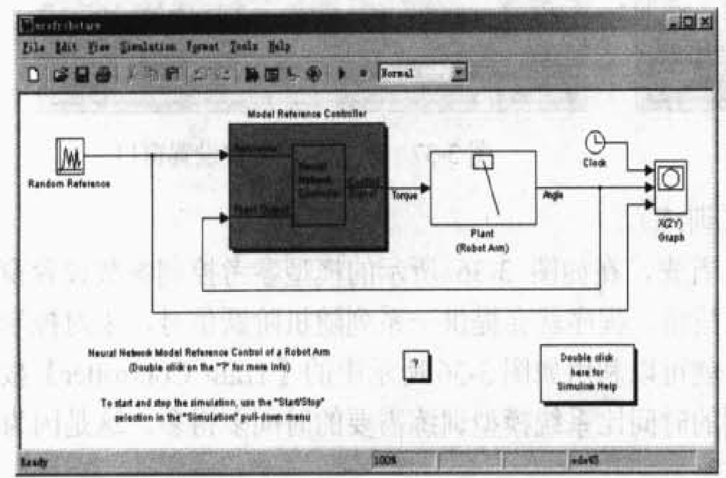


图 3-34　模型窗口

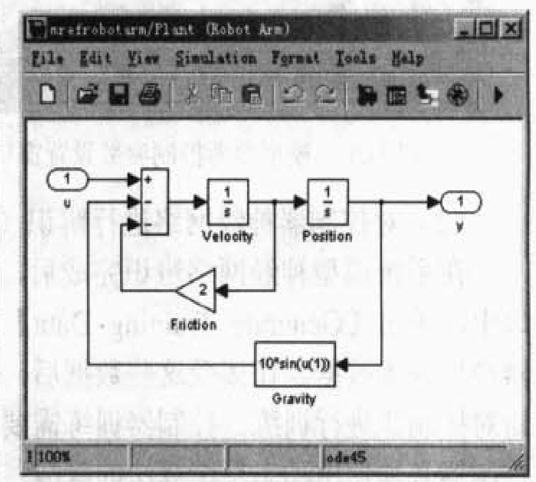


图 3-35　机械臂系统模型窗口

3）系统辨识

神经网络模型参考控制系统使用两个神经网络：一个控制器神经网络和一个系统模型神经网络。首先，对系统模型神经网络进行辨识，然后，对控制器神经网络进行辨识（训练），使得系统输出跟踪参考模型的输出。

（1）对系统模型神经网络进行辨识

在图 3-34 中，双击模型参考控制模块，将会产生一个模型参考控制参数（Mode Reference Control）设置窗口，如图 3-36 所示。这个窗口用于训练模型参考神经网络。窗口中各参数的设置说明参照前面的解释。

在如图 3-36 所示的模型参考控制窗口中，单击【Plant Identification】按钮，将会弹出一个如图 3-37 所示的系统辨识参数设置窗口。系统辨识过程的操作同前，在系统辨识结束后，单击图 3-37 中的【Accept Data】按钮，返回到模型参考控制窗口图 3-36 中。

图 3-36 模型参考控制参数设置窗口

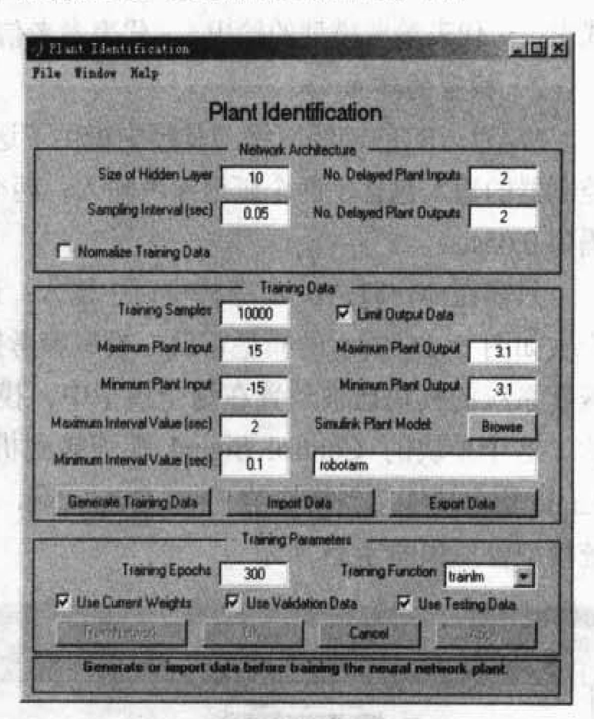


图 3-37 系统辨识参数设置窗口

（2）对控制器神经网络进行辨识（训练）

在系统模型神经网络辨识完成后，首先，在如图 3-36 所示的模型参考控制参数设置窗口中，单击【Generate Training Data】按钮，程序就会提供一系列随机阶跃信号，来对控制器产生训练数据。在接受这些数据后，就可以利用如图 3-36 所示中的【Train Controller】按钮对控制器进行训练。控制器训练需要的时间比系统模型训练需要的时间多得多。这是因为控制器必须使用动态反馈算法的缘故。

训练过程误差曲线如图 3-38 所示。训练结束后，返回到模型参考控制器窗口图 3-36 中，若控制器的性能不准确，则可以再次单击【Train Controller】按钮，这样就会继续使用同样的数据对控制器进行训练。若需要使用新的数据继续训练，则可以在单击【Train Controller】按钮之前再次单击【Generate Training Data】按钮或者【Import Data】按钮（注意，要确认 Use Current Weights 被选中）。另外，系统模型不够准确，也会影响控制器的训练。

在模型参考控制窗口图 3-36 中，单击【OK】按钮，将训练好的神经网络控制器权值导


入 Simulink 模型窗口，并返回到 Simulink 模型窗口图 3-34 中。

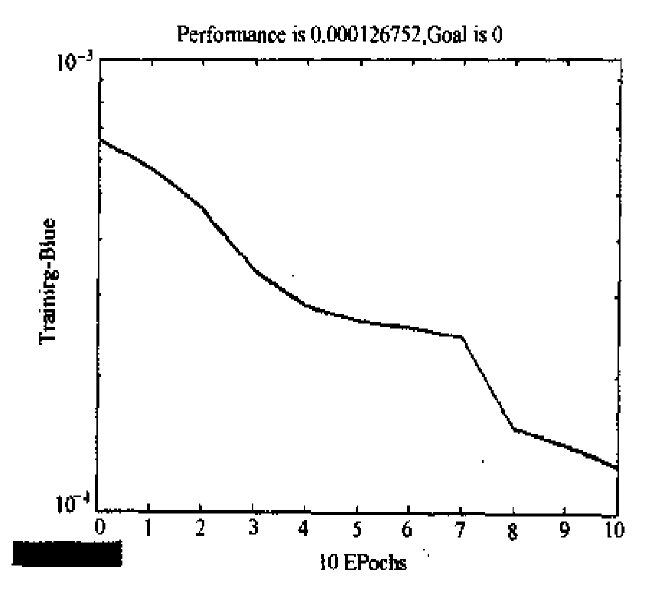


图 3-38　训练过程误差曲线

4）系统仿真

在 Simulink 模型窗口图 3-34 中，首先选择【Simulation】菜单中的【Parameter】命令，设置相应的仿真参数，然后从【Simulation】菜单中单击【Start】命令，开始仿真。仿真的过程需要一段时间。当仿真结束时，将会显示出系统的输出和参考信号，如图 3-39 所示。

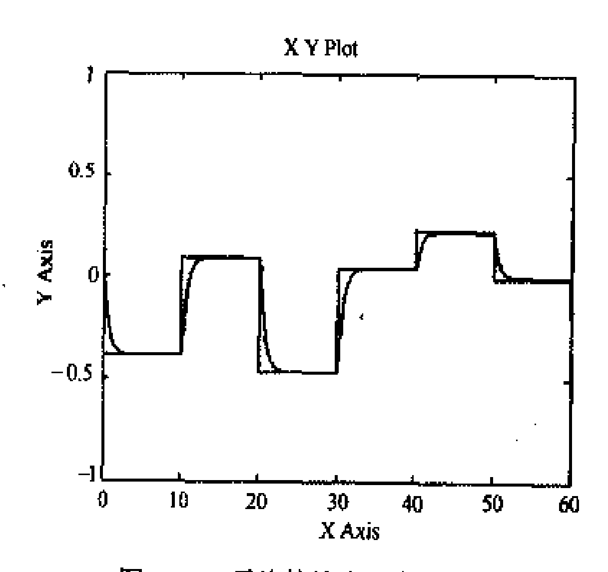


图 3-39　系统的输出和参考信号

对于模型参考控制，首先建立一个神经网络系统模型。接着，使用这个系统模型来训练一个神经网络控制器，迫使系统输出跟踪参考模型的输出。这种控制结构需要使用动态反传算法来训练控制器。在通常情况下，它比使用标准的反传算法训练静态网络花费的时间要多。然而，这种方法比 NARMA-L2（反馈线性化）控制结构更能适应一般的情况。这种控制器需要的在线计算时间最少。

第二篇　模糊逻辑控制及其MATLAB实现

第4章　模糊逻辑控制理论

控制论的创始人维纳教授在谈到人胜过最完善的机器时说："人具有运用模糊概念的能力"。这清楚地指明了人脑与电脑之间有着本质的区别，人脑具有善于判断和处理模糊现象的能力。"模糊"是与"精确"相对的概念。模糊性普遍存在于人类思维和语言交流中，是一种不确定性的表现。随机性则是客观存在的另一类不确定性，两者虽然都是不确定性，但存在本质的区别：模糊性主要是人对概念外延的主观理解上的不确定性；随机性则主要反映客观上的自然的不确定性，即对事件或行为的发生与否的不确定性。

模糊逻辑和模糊数学虽然只有短短的几十年历史，但其理论和应用的研究已取得了丰硕的成果。尤其是随着模糊逻辑在自动控制领域的成功应用，模糊控制理论和方法的研究引起了学术界和工业界的广泛关注。在模糊理论研究方面，以Zadeh提出的分解定理和扩张原则为基础的模糊数学理论已有大量的成果问世。1984年成立了国际模糊系统协会（IFSA），FUZZY SETS AND SYSTEMS（模糊集与系统）杂志与IEEE（美国电气与电子工程师协会）的"模糊系统"杂志也先后创刊。在模糊逻辑的应用方面，自从1974年英国的Mamdani首次将模糊逻辑用于蒸汽机的控制后，模糊控制在工业过程控制、机器人、交通运输等方面得到了广泛而卓有成效的应用。与传统控制方法，如PID控制相比，模糊控制利用人类专家控制经验，对于非线性、复杂对象的控制显示了鲁棒性好、控制性能高的优点。模糊逻辑的其他应用领域包括聚类分析、故障诊断、专家系统和图像识别等。

4.1　模糊逻辑理论的基本概念

4.1.1　模糊集合及其运算

集合一般指具有某种属性的、确定的、彼此间可以区别的事物的全体。将组成集合的事物称为集合的元素或元。通常用大写字母 $A,B,C,\cdots X,Y,Z$ 表示集合，而用小写字母 $a,b,c,\cdots,x,y,z$ 表示集合内元素。被考虑对象的所有元素的全体称为论域，一般用大写字母 U 表示。

在康托创立的经典集合论中，一事物要么属于某集合，要么不属于某集合，二者必居其一，没有模棱两可的情况，即经典集合所表达的概念的内涵和外延都必须是明确的。

在人们的思维中，有许多没有明确外延的概念，即模糊概念。在语言方面也有许多模糊概念的词，如果以人的年龄为论域，那么“年轻”、“中年”、“年老”都没有明确的外延。再如以某炉温为论域，那么“高温”、“中温”、“低温”等也都没有明确的外延。诸如此类的概念都是模糊概念。模糊概念不能用经典集合加以描述，原因是它不能绝对地用“属于”或“不属于”某集合来表示，也就是说，论域上的元素符合概念的程度不是绝对的 0 或 1，而是介于 0 和 1 之间的一个实数。

1．模糊集合的定义及表示方法

Zadeh 在 1965 年对模糊集合的定义为：给定论域 U,U 到[0,1]闭区间的任一映射 μ_A

$$\mu_A:\ U\to[0,1]$$

都确定 U 的一个模糊集合 A,μ_A 称为模糊集合 A 的隶属函数。它反映了模糊集合中的元素属于该集合的程度。若 A 中的元素用 x 表示，则 $\mu_A(x)$ 称为 x 属于 A 的隶属度。$\mu_A(x)$ 的取值范围为闭区间[0,1]，$\mu_A(x)$ 接近 1，表示 x 属于 A 的程度高；$\mu_A(x)$ 接近 0，表示 x 属于 A 的程度低。可见，模糊集合完全由隶属函数所描述。

模糊集合有很多表示方法，最常用的有以下几种。

当论域 U 为有限集 $\{x_1,x_2,\cdots,x_n\}$ 时，通常有以下三种方式：

1）Zadeh 表示法

用论域中的元素 x_i 与其隶属度 $\mu_A(x_i)$ 按下式表示 A，则

$$A=\frac{\mu_A(x_1)}{x_1}+\frac{\mu_A(x_2)}{x_2}+\cdots+\frac{\mu_A(x_n)}{x_n}$$

式中，$\mu_A(x_i)/x_i$ 并不表示“分数”，而是表示论域中的元素 x_i 与其隶属度 $\mu_A(x_i)$ 之间的对应关系；“+”也不表示“求和”，而是表示模糊集合在论域 U 上的整体。在 Zadeh 表示法中，隶属度为零的项可不写入。

2）序偶表示法

用论域中的元素 x_i 与其隶属度 $\mu_A(x_i)$ 构成序偶来表示 A，则

$$A=\{(x_1,\mu_A(x_1)),(x_2,\mu_A(x_2)),\cdots,(x_n,\mu_A(x_n))\mid x\in U\}$$

在序偶表示法中，隶属度为零的项可省略。

3）向量表示法

用论域中元素 x_i 的隶属度 $\mu_A(x_i)$ 构成向量来表示 A，则

$$A=[\mu_A(x_1)\ \mu_A(x_2)\ \cdots\ \mu_A(x_n)]$$

在向量表示法中，隶属度为零的项不能省略。

若 A 为以实数 $\mathbf{R}$ 为论域的模糊集合，其隶属函数为 $\mu_A(x)$，如果对任意实数 $a<x<b$，都有

$$\mu_A(x)\geqslant\min\{\mu_A(a),\mu_A(b)\}$$

则称 A 为凸模糊集。凸模糊集实质上就是隶属函数具有单峰值特性。今后所用的模糊集合一般均指凸模糊集。

例 4-1 在整数 1,2,…,10 组成的论域中，即论域 $U=\{1,2,\cdots,10\}$，用 A 表示模糊集合“几个”。并设各元素的隶属函数μ_A依次为{0,0,0.3,0.7,1,1,0.7,0.3,0,0}。

解：模糊集合 A 可表示为

$$A=\frac{0}{1}+\frac{0}{2}+\frac{0.3}{3}+\frac{0.7}{4}+\frac{1}{5}+\frac{1}{6}+\frac{0.7}{7}+\frac{0.3}{8}+\frac{0}{9}+\frac{0}{10}=\frac{0.3}{3}+\frac{0.7}{4}+\frac{1}{5}+\frac{1}{6}+\frac{0.7}{7}+\frac{0.3}{8}$$

$$A=\{(1,0),(2,0),(3,0.3),(4,0.7),(5,1),(6,1),(7,0.7),(8,0.3),(9,0),(10,0)\}=\{(3,0.3),(4,0.7),(5,1),(6,1),(7,0.7)\ (8,0.3)\}$$

$$A=[0\quad 0\quad 0.3\quad 0.7\quad 1\quad 1\quad 0.7\quad 0.3\quad 0\quad 0]$$

当论域 U 为有限连续域时，Zadeh 表示法为

$$A=\int_U\frac{\mu_A(x)}{x}$$

式中，$\mu_A(x)/x$ 也不表示“分数”，而是表示论域中的元素 x 与其隶属度$\mu_A(x)$之间的对应关系；“∫”也不表示“积分”，而是表示模糊集合在论域 U 上的元素 x 与其隶属度$\mu_A(x)$对应关系的一个整体。同样，在有限连续域表示法中，隶属度为零的部分可不写入。

例 4-2 若以年龄为论域，并设 $U=[0,200]$，设 Y 表示模糊集合“年轻”，O 表示模糊集合“年老”。已知“年轻”和“年老”的隶属函数分别为

$$\mu_Y(x)=\begin{cases}1 & 0\leqslant x\leqslant 25\\ \dfrac{1}{1+\left(\dfrac{x-25}{5}\right)^2} & 25<x\leqslant 200\end{cases}$$

$$\mu_O(x)=\begin{cases}0 & 0\leqslant x\leqslant 50\\ \dfrac{1}{1+\left(\dfrac{5}{x-50}\right)^2} & 50<x\leqslant 200\end{cases}$$

解：因为论域是连续的，因而“年轻”和“年老”的模糊集合 Y 和 O 分别为

$$Y=\{(x,1)\mid 0\leqslant x\leqslant 25\}+\left\{\left(x,\left[1+\left(\frac{x-25}{5}\right)^2\right]^{-1}\right)\middle| 25<x\leqslant 200\right\}$$

$$O=\{(x,0)\mid 0\leqslant x\leqslant 50\}+\left\{\left(x,\left[1+\left(\frac{5}{x-50}\right)^2\right]^{-1}\right)\middle| 50<x\leqslant 200\right\}$$

或

$$Y=\int_{0\leqslant x\leqslant 25}\frac{1}{x}+\int_{25<x\leqslant 200}\frac{\left[1+\left(\dfrac{x-25}{5}\right)^2\right]^{-1}}{x}$$

$$O=\int_{0\le x\le 50}\frac{0}{x}+\int_{50<x\le 200}\frac{\left[1+\left(\frac{5}{x-50}\right)^{2}\right]^{-1}}{x}$$

其隶属度函数曲线如图 4-1 所示。

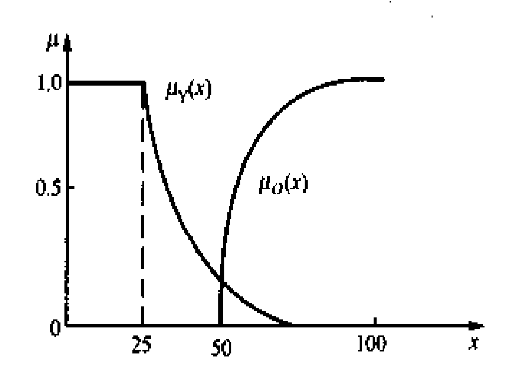


图 4-1　“年轻”和“年老”的隶属度函数曲线

2. 隶属函数

隶属函数是对模糊概念的定量描述，正确地确定隶属函数，是运用模糊集合理论解决实际问题的基础。隶属函数的确定过程，本质上说应该是客观的，但每个人对于同一个模糊概念的认识理解又有差异，因此，隶属函数的确定又带有主观性。它一般是根据经验或统计进行确定，也可由专家、权威给出。

以实数域 **R** 为论域时，称隶属函数为模糊分布。常见的模糊分布有以下四种

1）正态型

正态型是最主要也是最常见的一种分布，表示为

$$\mu(x)=\mathrm{e}^{-\left(\frac{x-a}{b}\right)^{2}}\quad b>0$$

其分布曲线如图 4-2 所示。

2）Γ型

$$\mu(x)=\begin{cases}0 & x<0\\ \left(\dfrac{x}{\lambda v}\right)^{v}\cdot \mathrm{e}^{v-\frac{x}{\lambda}} & x\geqslant 0\end{cases}$$

式中，$\lambda>0, v>0$。当 $x=\lambda v$ 时，隶属度函数为 1。其分布曲线如图 4-3 所示。

3）戒上型

$$\mu(x)=\begin{cases}\dfrac{1}{1+[a(x-c)]^{b}} & x>c\\ 1 & x\leqslant c\end{cases}$$

式中，$a>0, b>0$。其分布曲线如图 4-4 所示。

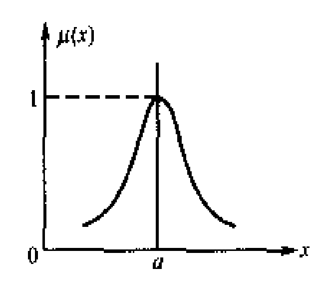


图 4-2 正态型分布曲线

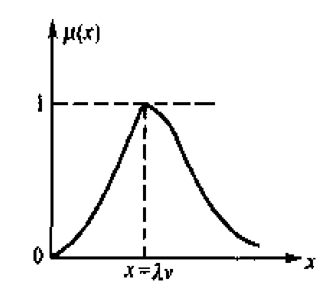


图 4-3 Γ型分布曲线

当 a=0.2, b=2, c=25 时，即为“年轻”的隶属函数。

4）戒下型

$$\mu(x)=\begin{cases}0 & x<c \\ \dfrac{1}{1+[a(x-c)]^b} & x\geqslant c\end{cases}$$

式中，a>0, b<0。其分布曲线如图 4-5 所示。

当 a=0.2, b= −2, c=50 时，即为“年老”的隶属函数。

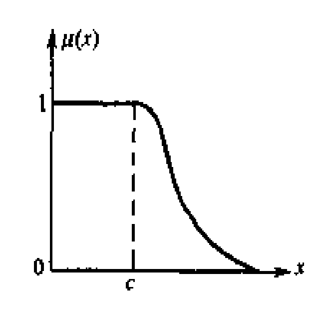


图 4-4 戒上型分布曲线

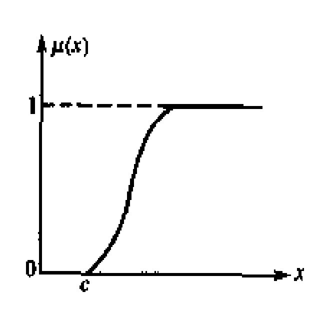


图 4-5 戒下型分布曲线

3. 模糊集合的有关术语

1）台集合

定义 $$A_s=\{x|\mu_A(x)>0\}$$

为 A 的台集合。其意义为论域 U 中所有使$\mu_A(x)>0$ 的 x 的全体。例 4-1 中，模糊集合 A 的台集合为

$$A_s=\{3,4,5,6,7,8\}$$

显然，台集合为普通集合，即

$$\mu_{A_s}(x)=\begin{cases}1 & x\in A_s \\ 0 & x\notin A_s\end{cases}$$

模糊集合只可在它的台集合上加以表示。

2）α 截集

定义 $$A_\alpha=\{x|\mu_A(x)>\alpha\},\alpha\in[0,1);A_{\overline{\alpha}}=\{x|\mu_A(x)\geqslant\alpha\},\alpha\in(0,1]$$

分别称为模糊集合 A 的强α 截集和弱α 截集。显然，α 截集也为普通集合，且 $A_s=A_\alpha|_{\alpha=0}$。

3）正则模糊集合

若

$$\max_{x\in X}\mu_A(x)=1$$

则称 A 为正则模糊集合。

4）凸模糊集合

若

$$\mu_A(\lambda x_1+(1-\lambda)x_2)\geqslant\min(\mu_A(x_1),\mu_A(x_2)),x_1,x_2\in U,\lambda\in[0,1]$$

则称 A 为凸模糊集合。

5）分界点

使得$\mu_A(x)=0.5$ 的点 x 称为模糊集合 A 的分界点。

6）单点模糊集合

在论域中，若模糊集合的台集合仅为一个点，且该点的隶属函数$\mu_A(x)=1$，则称 A 为单点模糊集合。

4. 分解定理和扩张原则

1）分解定理

设 A 为论域 U 上的一个模糊集合，A_α是 A 的α 截集，$\alpha\in[0, 1]$，则有如下分解定理成立，即

$$A=\bigcup_{\alpha\in[0,1]}\alpha A_\alpha$$

式中，αA_α表示语言变量 x 的一个模糊集合，称为α与 A_α的“乘积”。其隶属函数定义为

$$\mu_{\alpha A_\alpha}(x)=\begin{cases}\alpha,x\in A_\alpha\\0,x\notin A_\alpha\end{cases}$$

例 4-3　求模糊集合

$$A=\frac{0.5}{u_1}+\frac{0.6}{u_2}+\frac{1}{u_3}+\frac{0.7}{u_4}+\frac{0.3}{u_5}$$

的α 截集，$\alpha\in[0, 1]$。

解：取α分别为 1,0.7,0.6,0.5,0.3，于是有

$$A_1=\{u_3\},A_{0.7}=\{u_3,u_4\},A_{0.6}=\{u_2,u_3,u_4\}$$
$$A_{0.5}=\{u_1,u_2,u_3,u_4\},A_{0.3}=\{u_1,u_2,u_3,u_4,u_5\}$$

将α 截集写成模糊集合的形式

$$A_1=\frac{1}{u_3},A_{0.7}=\frac{1}{u_3}+\frac{1}{u_4},A_{0.6}=\frac{1}{u_2}+\frac{1}{u_3}+\frac{1}{u_4}$$
$$A_{0.5}=\frac{1}{u_1}+\frac{1}{u_2}+\frac{1}{u_3}+\frac{1}{u_4},A_{0.3}=\frac{1}{u_1}+\frac{1}{u_2}+\frac{1}{u_3}+\frac{1}{u_4}+\frac{1}{u_5}$$

则有

$$1A_1=\frac{1}{u_3},0.7A_{0.7}=\frac{0.7}{u_3}+\frac{0.7}{u_4},0.6A_{0.6}=\frac{0.6}{u_2}+\frac{0.6}{u_3}+\frac{0.6}{u_4}$$

$$0.5A_{0.5}=\frac{0.5}{u_1}+\frac{0.5}{u_2}+\frac{0.5}{u_3}+\frac{0.5}{u_4},0.3A_{0.3}=\frac{0.3}{u_1}+\frac{0.3}{u_2}+\frac{0.3}{u_3}+\frac{0.3}{u_4}+\frac{0.3}{u_5}$$

由分解定理又可构成原来的模糊集合

$$\bigcup_{\alpha\in[0,1]}\alpha A_\alpha=1A_1+0.7A_{0.7}+0.6A_{0.6}+0.5A_{0.5}+0.3A_{0.3}$$

$$=\frac{1}{u_3}\cup\left(\frac{0.7}{u_3}+\frac{0.7}{u_4}\right)\cup\left(\frac{0.6}{u_2}+\frac{0.6}{u_3}+\frac{0.6}{u_4}\right)\cup\left(\frac{0.5}{u_1}+\frac{0.5}{u_2}+\frac{0.5}{u_3}+\frac{0.5}{u_4}\right)\cup$$
$$\left(\frac{0.3}{u_1}+\frac{0.3}{u_2}+\frac{0.3}{u_3}+\frac{0.3}{u_4}+\frac{0.3}{u_5}\right)$$

$$=\frac{0.3\vee0.5}{u_1}+\frac{0.3\vee0.5\vee0.6}{u_2}+\frac{0.3\vee0.5\vee0.6\vee0.7\vee1}{u_3}+\frac{0.3\vee0.5\vee0.6\vee0.7}{u_4}+\frac{0.3}{u_5}$$

$$=\frac{0.5}{u_1}+\frac{0.6}{u_2}+\frac{1}{u_3}+\frac{0.7}{u_4}+\frac{0.3}{u_5}=A$$

2）扩张原则

设 U 和 V 是两个论域，f 是 U 到 V 的一个映射，对 U 上的模糊集合 A，可以扩张成为

$$\tilde{f}:A=\tilde{f}(A)$$

这里，$\tilde{f}$ 叫做 f 的扩张。A 通过 $\tilde{f}$ 映射成 $\tilde{f}(A)$ 时，规定它的隶属函数的值保持不变。在不会误解的情况下，$\tilde{f}$ 可以称为 f。

分解定理和扩张原则是模糊数学的理论支柱。分解定理是联系模糊数学和普通数学的纽带，而扩张原则是把普通的数学扩展到模糊数学的有力工具。

5．模糊集合的运算

1）模糊集合的相等

若有两个模糊集合 A 和 B，对于所有的 $x\in U$，均有 $\mu_A(x)=\mu_B(x)$，则称模糊集合 A 等于模糊集合 B，记为 $A=B$。

2）模糊集合的包含关系

若有两个模糊集合 A 和 B，对于所有的 $x\in U$，均有 $\mu_A(x)\leqslant\mu_B(x)$，则称模糊集合 A 包含于模糊集合 B，或 A 是 B 的子集，记为 $A\subseteq B$。

3）模糊空集

若对于所有的 $x\in U$，均有 $\mu_A(x)=0$，则称模糊集合 A 为空集，记为 $A=\Phi$。

4）模糊集合的并集

若有三个模糊集合 A、B 和 C，对于所有的 $x\in U$，均有

$$\mu_C(x)=\mu_A(x)\vee\mu_B(x)=\max[\mu_A(x),\mu_B(x)]$$

则称模糊集合 C 为 A 与 B 的并集，记为 $C=A\cup B$。

5）模糊集合的交集

若有三个模糊集合 A、B 和 C，对于所有的 $x\in U$，均有

$$\mu_C(x)=\mu_A(x)\wedge\mu_B(x)=\min[\mu_A(x),\mu_B(x)]$$

则称模糊集合 C 为 A 与 B 的交集，记为 $C=A\cap B$。

6）模糊集合的补集

若有两个模糊集合 A 和 B，对于所有的 $x\in U$，均有$\mu_B(x)=1-\mu_A(x)$，则称 B 为 A 的补集，记为 $B=A^c$。

7）模糊集合的直积

若有两个模糊集合 A 和 B，其论域分别为 X 和 Y，则定义在积空间 $X\times Y$ 上的模糊集合 $A\times B$ 称为模糊集合 A 和 B 的直积，即

$$A\times B=\{(a,b)|a\in A,b\in B\}$$

上述定义表明，在集合 A 中取一元素 a，又在集合 B 中取一元素 b，就构成了(a,b)"序偶"，所有的(a,b)又构成一个集合。该集合即为 $A\times B$。其隶属函数为

$$\mu_{A\times B}(x,y)=\min[\mu_A(x),\mu_B(y)]$$

或者

$$\mu_{A\times B}(x,y)=\mu_A(x)\mu_B(y)$$

直积又称为笛卡儿积或叉积。两个模糊集合直积的概念可以很容易推广到多个集合。

若 $\mathbf{R}$ 是实数集，即 $\mathbf{R}=\{x|-\infty<x<+\infty\}$，则 $\mathbf{R}\times\mathbf{R}=\{(x,y)|-\infty<x<+\infty,-\infty<y<+\infty\}$，用 $\mathbf{R}^2$ 表示，$\mathbf{R}^2=\mathbf{R}\times\mathbf{R}$ 即为整个平面，这就是二维欧氏空间。同理，$\mathbf{R}\times\mathbf{R}\times\cdots\times\mathbf{R}=\mathbf{R}^n$ 称为 n 维欧氏空间。

6. 模糊集合运算的基本性质

（1）幂等律：$A\cup A=A,A\cap A=A$

（2）交换律：$A\cup B=B\cup A,A\cap B=B\cap A$

（3）结合律：$(A\cap B)\cap C=A\cap(B\cap C),(A\cup B)\cup C=A\cup(B\cup C)$

（4）分配律：$A\cap(B\cup C)=(A\cap B)\cup(A\cap C),A\cup(B\cap C)=(A\cup B)\cap(A\cup C)$

（5）吸收律：$(A\cap B)\cup A=A,(A\cup B)\cap A=A$

（6）同一律：$A\cup\Omega=\Omega,A\cap\Omega=A,A\cup\Phi=A,A\cap\Phi=\Phi$，其中，$\Omega$表示全集，$\Phi$表示空集。

（7）复原律：$(A^c)^c=A$

（8）对偶律：$(A\cup B)^c=A^c\cap B^c,(A\cap B)^c=A^c\cup B^c$

7. 模糊集合的其他类型运算

（1）代数和：$A\dot{+}B\leftrightarrow\mu_{A\dot{+}B}(x)=\mu_A(x)+\mu_B(x)-\mu_A(x)\mu_B(x)$

（2）代数积：$A\cdot B\leftrightarrow\mu_{A\cdot B}(x)=\mu_A(x)\mu_B(x)$

（3）有界和：$A\oplus B\leftrightarrow\mu_{A\oplus B}(x)=\min\{1,\mu_A(x)+\mu_B(x)\}$

（4）有界差：$A \ominus B \leftrightarrow \mu_{A \ominus B}(x) = \max\{0, \mu_A(x) - \mu_B(x)\}$

（5）有界积：$A \odot B \leftrightarrow \mu_{A \odot B}(x) = \max\{0, \mu_A(x) + \mu_B(x) - 1\}$

（6）强制和(drastic sum)：

$$A \dot{\cup} B \leftrightarrow \mu_{A \dot{\cup} B}(x) = \begin{cases} \mu_A(x), \mu_B(x) = 0 \\ \mu_B(x), \mu_A(x) = 0 \\ 1, \mu_A(x), \mu_B(x) > 0 \end{cases}$$

（7）强制积(drastic product)：

$$A \dot{\cap} B \leftrightarrow \mu_{A \dot{\cap} B}(x) = \begin{cases} \mu_A(x), \mu_B(x) = 1 \\ \mu_B(x), \mu_A(x) = 1 \\ 0, \mu_A(x), \mu_B(x) < 1 \end{cases}$$

4.1.2 模糊关系及其合成

在日常生活中，经常会听到诸如“A 与 B 很相似”、“X 比 Y 大得多”等描述模糊关系的语句。模糊关系在模糊集合论中占有重要的地位，当论域为有限时，可以用模糊矩阵来表示模糊关系。

1. 模糊关系

设 X、Y 是两个非空集合，则在直积

$$X \times Y = \{(x, y) | x \in X, y \in Y\}$$

中一个模糊集合 R 称为从 X 到 Y 的一个模糊关系，记为 R_{xy}。

模糊关系 R_{xy} 由其隶属函数 $\mu_R(x, y)$ 完全刻画，$\mu_R(x, y)$ 表示了 X 中的元素 x 与 Y 中的元素 y 具有关系 R_{xy} 的程度。

以上定义的模糊关系又称二元模糊关系，当 $X=Y$ 时，称为 X 上的模糊关系。

当论域为 n 个集合的直积

$$X_1 \times X_2 \times \cdots \times X_n = \{(x_1, x_2, \cdots, x_n) | x_i \in X_i, i=1,2,\cdots,n\}$$

时，它所对应的为 n 元模糊关系 $R_{x_1 \times x_2 \times \cdots \times x_n}$。

当论域 $X=\{x_1, x_2, \cdots, x_n\}$，$Y=\{y_1, y_2, \cdots, y_m\}$ 是有限集合时，定义在 $X \times Y$ 上的模糊关系 R_{xy} 可用如下的 $n \times m$ 阶矩阵来表示，即

$$R = \begin{bmatrix} \mu_R(x_1, y_1) & \mu_R(x_1, y_2) & \cdots & \mu_R(x_1, y_m) \\ \mu_R(x_2, y_1) & \mu_R(x_2, y_2) & \cdots & \mu_R(x_2, y_m) \\ \vdots & \vdots & \vdots & \vdots \\ \mu_R(x_n, y_1) & \mu_R(x_n, y_2) & \cdots & \mu_R(x_n, y_m) \end{bmatrix}$$

这样的矩阵称为模糊矩阵。模糊矩阵 R 中元素 $r_{ij}=\mu_R(x_i, y_j)$ 表示论域 X 中第 i 个元素 x_i 与论域 Y 中的第 j 个元素 y_j 对于模糊关系 R_{xy} 的隶属程度，因此它们均在[0，1]中取值。

由于模糊关系是定义在直积空间上的模糊集合，因此它也遵从一般模糊集合的运算规则。

例 4-4　设 X 为家庭中的儿子和女儿，Y 为家庭中的父亲和母亲，对于“子女与父母长得相似”的模糊关系 R，可以用以下模糊矩阵 R 表示。

$$\boldsymbol{R}=\begin{matrix} & \text{父} & \text{母} \\ \text{子} & 0.8 & 0.3 \\ \text{女} & 0.3 & 0.6 \end{matrix}$$

2. 模糊关系的合成

设 X、Y、Z 是论域，$R_{x\times y}$ 是 X 到 Y 的一个模糊关系，$S_{y\times z}$ 是 Y 到 Z 的一个模糊关系，则 $R_{x\times y}$ 到 $S_{y\times z}$ 的合成 $T_{x\times z}$ 也是一个模糊关系，记为

$$T_{x\times z}=R_{x\times y}\circ S_{y\times z}$$

它具有隶属度

$$\mu_{R\circ S}(x,z)=\bigvee_{y\in Y}(\mu_R(x,y)*\mu_S(y,z))$$

式中，“$\vee$”是并的符号，它表示对所有 y 取极大值或上界值；“$*$”是二项积的符号，因此上面的合成称为最大–星合成(max-star composition)。

二项积算子“$x*y$”可以定义为以下几种运算，其中 $x,y\in[0,1]$。

（1）交：$x*y=x\wedge y=\min\{x,y\}$

（2）代数积：$x*y=x\cdot y=xy$

（3）有界积：$x*y=x\odot y=\max\{0,x+y-1\}$

当二项积算子“$*$”采用前两种运算时，它们分别称为最大–最小合成和最大–积合成，即

$$\mu_{R\circ S}(x,z)=\bigvee_{y\in Y}(\mu_R(x,y)\wedge\mu_S(y,z))$$

或

$$\mu_{R\circ S}(x,z)=\bigvee_{y\in Y}(\mu_R(x,y)\mu_S(y,z))$$

其中最大–最小合成最为常用。以后如无特别说明，则均指此合成。

当论域 X、Y、Z 为有限时，模糊关系的合成可用模糊矩阵来表示。设 $R_{x\times y}$、$S_{y\times z}$、$T_{x\times z}$ 三个模糊关系对应的模糊矩阵分别为

$$\boldsymbol{R}=(r_{ij})_{n\times m},\boldsymbol{S}=(s_{jk})_{m\times l},\boldsymbol{T}=(t_{ik})_{n\times l}$$

则

$$t_{ik}=\bigvee_{j=1}^{m}(r_{ij}\wedge s_{jk})\ \text{或}\ t_{ik}=\bigvee_{j=1}^{m}(r_{ij}\cdot s_{jk})\quad(i=1,2,\cdots,n;k=1,2,\cdots,l)$$

即用模糊矩阵的合成 $T=R\circ S$ 来表示模糊关系的合成 $T_{x\times z}=R_{x\times y}\circ S_{y\times z}$。

例 4-5　已知子女与父母相似关系的模糊矩阵 R 和父母与祖父母相似关系的模糊矩阵 S 分别如下所示，求子女与祖父母的相似关系模糊矩阵。

$$R=\begin{matrix} & \text{父} & \text{母} \\ \text{子} & 0.8 & 0.3 \\ \text{女} & 0.3 & 0.6\end{matrix},\quad S=\begin{matrix} & \text{祖父} & \text{祖母} \\ \text{父} & 0.7 & 0.5 \\ \text{母} & 0.1 & 0.1\end{matrix}$$

解：这是一个典型的模糊关系合成的问题。按最大-最小合成规则有

$$T=R\circ S=\begin{bmatrix}0.8 & 0.3\\0.3 & 0.6\end{bmatrix}\circ\begin{bmatrix}0.7 & 0.5\\0.1 & 0.1\end{bmatrix}=\begin{bmatrix}(0.8\wedge 0.7)\vee(0.3\wedge 0.1) & (0.8\wedge 0.5)\vee(0.3\wedge 0.1)\\(0.3\wedge 0.7)\vee(0.6\wedge 0.1) & (0.3\wedge 0.5)\vee(0.6\wedge 0.1)\end{bmatrix}$$

$$=\begin{bmatrix}0.7\vee 0.1 & 0.5\vee 0.1\\0.3\vee 0.1 & 0.3\vee 0.1\end{bmatrix}=\begin{matrix} & \text{祖父} & \text{祖母} \\ \text{子} & 0.7 & 0.5 \\ \text{女} & 0.3 & 0.3\end{matrix}$$

4.1.3 模糊向量及其运算

1. 模糊向量

如果对任意的 $i\,(i=1,2,\cdots,n)$ 都有 $x_i\in[0,1]$，则称向量

$$\boldsymbol{x}=[x_1,x_2,\cdots,x_n]$$

为模糊向量。

2. 模糊向量的笛卡儿乘积

设有 $1\times n$ 维模糊向量 $\boldsymbol{x}$ 和 $1\times m$ 维模糊向量 $\boldsymbol{y}$，则定义

$$\boldsymbol{x}\times\boldsymbol{y}\equiv\boldsymbol{x}^{\mathrm{T}}\circ\boldsymbol{y}$$

为模糊向量 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 的笛卡儿乘积。模糊向量 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 的笛卡儿乘积表示它们所在论域 $\boldsymbol{X}$ 与 $\boldsymbol{Y}$ 之间的转换关系，这种转换关系也是模糊关系，而上式右端正是模糊关系的合成运算。

例 4-6 已知两个模糊向量分别如下所示，求它们的笛卡儿乘积。

$$\boldsymbol{x}=[0.8\ 0.6\ 0.2],\quad \boldsymbol{y}=[0.2\ 0.4\ 0.7\ 1]$$

解：笛卡儿乘积为

$$\boldsymbol{x}\times\boldsymbol{y}=\boldsymbol{x}^{\mathrm{T}}\circ\boldsymbol{y}=\begin{bmatrix}0.8\\0.6\\0.2\end{bmatrix}\circ[0.2\quad 0.4\quad 0.7\quad 1]$$

$$=\begin{bmatrix}0.8\wedge 0.2 & 0.8\wedge 0.4 & 0.8\wedge 0.7 & 0.8\wedge 1\\0.6\wedge 0.2 & 0.6\wedge 0.4 & 0.6\wedge 0.7 & 0.6\wedge 1\\0.2\wedge 0.2 & 0.2\wedge 0.4 & 0.2\wedge 0.7 & 0.2\wedge 1\end{bmatrix}=\begin{bmatrix}0.2 & 0.4 & 0.7 & 0.8\\0.2 & 0.4 & 0.6 & 0.6\\0.2 & 0.2 & 0.2 & 0.2\end{bmatrix}$$

3. 模糊向量的内积与外积

设有 $1\times n$ 维模糊向量 $\boldsymbol{x}$ 和 $1\times n$ 维模糊向量 $\boldsymbol{y}$，则定义

$$\boldsymbol{x}\cdot\boldsymbol{y}\equiv\boldsymbol{x}\circ\boldsymbol{y}^{\mathrm{T}}=\bigvee_{i=1}^{n}(x_i\wedge y_i)$$

为模糊向量 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 的内积。与内积的对偶运算称为外积。

4.1.4　模糊逻辑规则

1. 模糊语言变量

语言是人们进行思维和信息交流的重要工具。语言可分为两种，即自然语言和形式语言。人们日常所用的语言属自然语言，它的特点是语义丰富、灵活。通常的计算机语言是形式语言，它只是形式上起记号作用。自然语言和形式语言最重要的区别在于，自然语言具有模糊性，而形式语言不具有模糊性，它完全具有二值逻辑的特点。

模糊语言变量是自然语言中的词或句，它的取值不是通常的数，而是用模糊语言表示的模糊集合。在不引起混淆的情况下，以下将模糊语言变量简称为语言变量。

一个语言变量可由以下的五元体来表征

$$(x,T(x),U,G,M)$$

式中，x 是语言变量的名称；$T(x)$是语言变量值的集合；U 是 x 的论域；G 是语法规则，用于产生语言变量 x 的名称；M 是语义规则，用于产生模糊集合的隶属函数。

例如，以控制系统的“误差”为语言变量 x，论域取 U=[−6,+6]。“误差”语言变量的原子单词有“大、中、小、零”，对这些原子单词施加以适当的语气算子，就可以构成多个语言值名称，如“很大”等，再考虑误差有正负的情况，$T(x)$可表示为

$T(x)$=T(误差)={负很大，负大，负中，负小，零，正小，正中，正大，正很大}

图 4-6 是误差语言变量的五元体示意图。

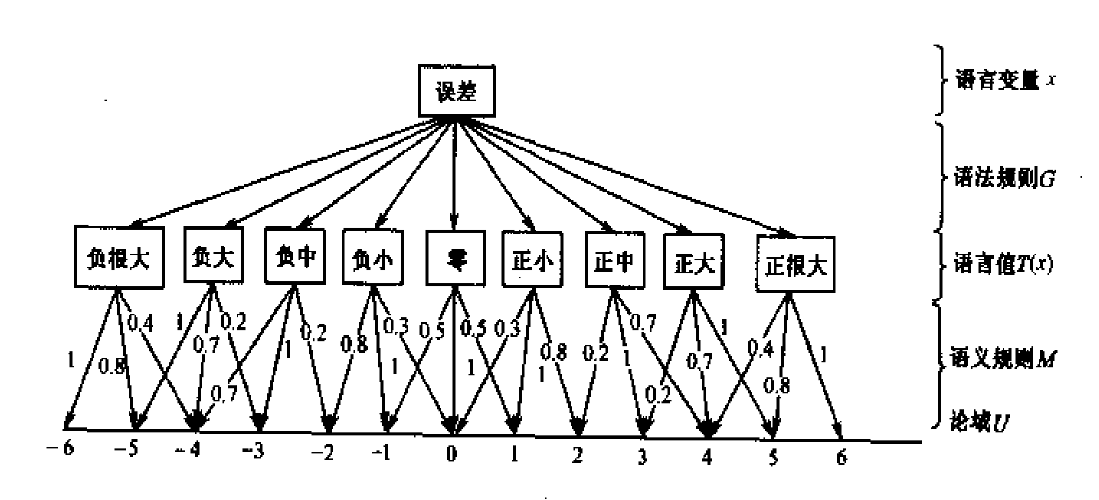


图 4-6　误差语言变量的五元体示意图

如上所述，每个模糊语言相当于一个模糊集合，在模糊语言前面加上“极”、“非常”、“相当”、“比较”、“略”、“稍微”、“非”等语气算子后，将改变该模糊语言的含义，相应地隶属函数也要改变。例如，设原来的模糊语言为 A，其隶属函数为μ_A，则通常有

$$\mu_{极A}=\mu_A^{4},\ \mu_{非常A}=\mu_A^{2},\ \mu_{相当A}=\mu_A^{1.25},\ \mu_{比较A}=\mu_A^{0.75},\ \mu_{略A}=\mu_A^{0.5},\ \mu_{稍微A}=\mu_A^{0.25},\ \mu_{非A}=1-\mu_A$$

2. 模糊蕴含关系

在模糊逻辑中，模糊逻辑规则实质上是模糊蕴含关系。在模糊逻辑推理中有很多定义模糊蕴含的方法，最常用的一类模糊蕴含关系是广义的肯定式推理方式，即

输入：如果 x 是 A'

前提：如果 x 是 A，则 y 是 B

结论：y 是 B'

此处，A,A',B,B' 均为模糊语言。横线上方是输入和前提条件，横线下方是结论。

对于模糊前提“如果 x 是 A，则 y 是 B”，它表示了模糊语言 A 与 B 之间的模糊蕴含关系，记为

$$A \to B$$

在普通的形式逻辑中，$A \to B$ 有严格的定义。但在模糊逻辑中，$A \to B$ 不是普通逻辑的简单推广，有许多定义的方法。在模糊逻辑控制中，常用的模糊蕴含关系的运算方法有以下几种，其中前两种最常用。

1）模糊蕴含的最小运算(Mamdani)

$$R_c = A \to B = A \times B = \int_{X \times Y} \mu_A(x) \wedge \mu_B(y) / (x, y)$$

2）模糊蕴含的积运算(Larsen)

$$R_p = A \to B = A \times B = \int_{X \times Y} \mu_A(x)\mu_B(y) / (x, y)$$

3）模糊蕴含的算术运算(Zadeh)

$$R_a = A \to B = (A^c \times Y) \oplus (X \times B) = \int_{X \times Y} 1 \wedge (1 - \mu_A(x) + \mu_B(y)) / (x, y)$$

4）模糊蕴含的最大最小运算(Zadeh)

$$R_m = A \to B = (A \times B) \cup (A^c \times Y) = \int_{X \times Y} (\mu_A(x) \wedge \mu_B(y)) \vee (1 - \mu_A(x)) / (x, y)$$

5）模糊蕴含的布尔运算

$$R_b = A \to B = (A^c \times Y) \cup (X \times B) = \int_{X \times Y} (1 - \mu_A(x)) \vee \mu_B(y) / (x, y)$$

6）模糊蕴含的标准法运算(1)

$$R_s = A \to B = A \times Y \to X \times B = \int_{X \times Y} (\mu_A(x) > \mu_B(y)) / (x, y)$$

其中 $\mu_A(x) > \mu_B(y) = \begin{cases} 1 & \mu_A(x) \leqslant \mu_B(y) \\ 0 & \mu_A(x) > \mu_B(y) \end{cases}$

7）模糊蕴含的标准法运算(2)

$$R_\Delta = A \to B = A \times Y \to X \times B = \int_{X \times Y} (\mu_A(x) >> \mu_B(y)) / (x, y)$$

其中　$\mu_A(x) >> \mu_B(y) = \begin{cases} 1 & \mu_A(x) \leqslant \mu_B(y) \\ \dfrac{\mu_B(y)}{\mu_A(x)} & \mu_A(x) > \mu_B(y) \end{cases}$

4.1.5　模糊逻辑推理

1. 简单模糊条件语句

对于上面介绍的广义肯定式推理，结论 $\boldsymbol{B}'$是根据模糊集合 $\boldsymbol{A}'$和模糊蕴含关系 $\boldsymbol{A} \to \boldsymbol{B}$ 的合成推出来的，因此可得如下的模糊推理关系

$$\boldsymbol{B}' = \boldsymbol{A}' \circ (\boldsymbol{A} \to \boldsymbol{B}) = \boldsymbol{A}' \circ \boldsymbol{R}$$

式中，$\boldsymbol{R}$ 为模糊蕴含关系，“o”是合成运算符。它们可采用以上所列举的任何一种运算方法。

例 4-7　若人工调节炉温，有如下的经验规则：“如果炉温低，则应施加高电压”。试问当炉温为“非常低”时，应施加怎样的电压。

解：设 x 和 y 分别表示模糊语言变量“炉温”和“电压”，并设 x 和 y 的论域为

$$X = Y = \{1,2,3,4,5\}$$

$\boldsymbol{A}$ 表示炉温低的模糊集合

$$\boldsymbol{A} = \text{“炉温低”} = \frac{1}{1} + \frac{0.8}{2} + \frac{0.6}{3} + \frac{0.4}{4} + \frac{0.2}{5}$$

$\boldsymbol{B}$ 表示高电压的模糊集合

$$\boldsymbol{B} = \text{“高电压”} = \frac{0.2}{1} + \frac{0.4}{2} + \frac{0.6}{3} + \frac{0.8}{4} + \frac{1}{5}$$

从而模糊规则可表述为：“如果 x 是 $\boldsymbol{A}$，则 y 是 $\boldsymbol{B}$”。设 $\boldsymbol{A}'$为非常 $\boldsymbol{A}$，则上述问题变为：“如果 x 是 $\boldsymbol{A}'$，则 $\boldsymbol{B}'$应是什么”。为了便于计算，将模糊集合 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 写成向量形式

$$\boldsymbol{A} = [1\ 0.8\ 0.6\ 0.4\ 0.2],\ \boldsymbol{B} = [0.2\ 0.4\ 0.6\ 0.8\ 1]$$

由于该例中 x 和 y 的论域均是离散的，因而模糊蕴含关系 $\boldsymbol{R}_c$ 可用如下模糊矩阵来表示

$$\boldsymbol{R}_c = \boldsymbol{A} \to \boldsymbol{B} = \boldsymbol{A} \times \boldsymbol{B} = \boldsymbol{A}^{\mathrm{T}} \circ \boldsymbol{B} = [1\quad 0.8\quad 0.6\quad 0.4\quad 0.2]^{\mathrm{T}} \cdot [0.2\quad 0.4\quad 0.6\quad 0.8\quad 1]$$

$$= \begin{bmatrix} 1 \wedge 0.2 & 1 \wedge 0.4 & 1 \wedge 0.6 & 1 \wedge 0.8 & 1 \wedge 1 \\ 0.8 \wedge 0.2 & 0.8 \wedge 0.4 & 0.8 \wedge 0.6 & 0.8 \wedge 0.8 & 0.8 \wedge 1 \\ 0.6 \wedge 0.2 & 0.6 \wedge 0.4 & 0.6 \wedge 0.6 & 0.6 \wedge 0.8 & 0.6 \wedge 1 \\ 0.4 \wedge 0.2 & 0.4 \wedge 0.4 & 0.4 \wedge 0.6 & 0.4 \wedge 0.8 & 0.4 \wedge 1 \\ 0.2 \wedge 0.2 & 0.2 \wedge 0.4 & 0.2 \wedge 0.6 & 0.2 \wedge 0.8 & 0.2 \wedge 1 \end{bmatrix} = \begin{bmatrix} 0.2 & 0.4 & 0.6 & 0.8 & 1 \\ 0.2 & 0.4 & 0.6 & 0.8 & 0.8 \\ 0.2 & 0.4 & 0.6 & 0.6 & 0.6 \\ 0.2 & 0.4 & 0.4 & 0.4 & 0.4 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \end{bmatrix}$$

当 $\boldsymbol{A}' = \text{“炉温非常低”} = \boldsymbol{A}^2 = [1\ 0.64\ 0.36\ 0.16\ 0.04]$时，

$$B' = A' \circ R_c = \begin{bmatrix}1 & 0.64 & 0.36 & 0.16 & 0.04\end{bmatrix} \circ \begin{bmatrix}0.2 & 0.4 & 0.6 & 0.8 & 1 \\ 0.2 & 0.4 & 0.6 & 0.8 & 0.8 \\ 0.2 & 0.4 & 0.6 & 0.6 & 0.6 \\ 0.2 & 0.4 & 0.4 & 0.4 & 0.4 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2\end{bmatrix}$$

$$= \begin{bmatrix}0.2 & 0.4 & 0.6 & 0.8 & 1\end{bmatrix}$$

其中$\boldsymbol{B}'$中的每项元素是根据模糊矩阵的合成规则求出的，如第1行第1列的元素为

$$0.2 = (1 \wedge 0.2) \vee (0.64 \wedge 0.2) \vee (0.36 \wedge 0.2) \vee (0.16 \wedge 0.2) \vee (0.04 \wedge 0.2)$$
$$= 0.2 \vee 0.2 \vee 0.2 \vee 0.16 \vee 0.04$$

这时，推论结果$\boldsymbol{B}'$仍为“高电压”。

2. 多重模糊条件语句

1）使用“and”连接的模糊条件语句

在模糊逻辑控制中，常常使用如下的广义肯定式推理方式

输入：如果 x 是 $\boldsymbol{A}'$ and y 是 $\boldsymbol{B}'$

前提：如果 x 是 $\boldsymbol{A}$ and y 是 $\boldsymbol{B}$，则 z 是 $\boldsymbol{C}$

结论：z 是 $\boldsymbol{C}'$

与前面不同的是，这里的模糊条件的输入和前提部分是将模糊命题用“and”连接起来的。一般情况下可以有多个“and”将多个模糊命题连接在一起。

模糊前提“x 是 $\boldsymbol{A}$，则 y 是 $\boldsymbol{B}$”可以看成是直积空间 $\boldsymbol{X}\times\boldsymbol{Y}$ 上的模糊集合，并记为 $\boldsymbol{A}\times\boldsymbol{B}$，其隶属函数为

$$\mu_{A\times B}(x,y)=\min\{\mu_A(x),\mu_B(y)\}$$

或者

$$\mu_{A\times B}(x,y)=\mu_A(x)\mu_B(y)$$

这时的模糊蕴含关系可记为 $\boldsymbol{A}\times\boldsymbol{B}\to\boldsymbol{C}$，其具体运算方法一般采用以下关系

$$\boldsymbol{R} = \boldsymbol{A}\times\boldsymbol{B}\to\boldsymbol{C} = \boldsymbol{A}\times\boldsymbol{B}\times\boldsymbol{C} = \int_{X\times Y\times Z} \mu_A(x)\wedge\mu_B(y)\wedge\mu_C(z)/(x,y,z)$$

结论 z 是 $\boldsymbol{C}'$，可根据如下的模糊推理关系得到

$$\boldsymbol{C}' = (\boldsymbol{A}'\times\boldsymbol{B}')\circ(\boldsymbol{A}\times\boldsymbol{B}\to\boldsymbol{C}) = (\boldsymbol{A}'\times\boldsymbol{B}')\circ\boldsymbol{R}$$

式中，$\boldsymbol{R}$ 为模糊蕴含关系；“∘”是合成运算符。它们可采用以上列举的任何一种运算方法。

2）使用“also”连接的模糊条件语句

在模糊逻辑控制中，也常常给出如下一系列的模糊控制规则

输入：如果 x 是 $\boldsymbol{A}'$ and y 是 $\boldsymbol{B}'$

前提1：如果 x 是 $\boldsymbol{A}_1$ and y 是 $\boldsymbol{B}_1$，则 z 是 $\boldsymbol{C}_1$

also 前提2：如果 x 是 $\boldsymbol{A}_2$ and y 是 $\boldsymbol{B}_2$，则 z 是 $\boldsymbol{C}_2$

…

also 前提 n：如果 x 是 A_n and y 是 B_n，则 z 是 C_n

输出：z 是 C'

这些规则之间无先后次序之分。连接这些子规则的连接词用“also”表示。这就要求对于“also”的运算具有能够任意交换和任意结合的性质。而求并和求交运算均能满足这样的要求。根据 Mizumoto 的研究结果，当模糊蕴含运算采用 R_c 或 R_p，“also”采用求并运算时，可得最好的控制结果。

假设第 i 条规则“如果 x 是 A_i and y 是 B_i，则 z 是 C_i”的模糊蕴含关系 R_i 定义为

$$R_i=(A_i \text{ and } B_i)\rightarrow C_i$$

其中“A_i and B_i”是定义在 $X\times Y$ 上的模糊集合 $A_i\times B_i$，$R_i=(A_i \text{ and } B_i)\rightarrow C_i$ 是定义在 $X\times Y\times Z$ 上的模糊蕴含关系。

则所有 n 条模糊控制规则的总模糊蕴含关系为（取连接词“also”为求并运算）

$$R=\bigcup_{i=1}^{n} R_i$$

输出模糊量 z（用模糊集合 C' 表示）为

$$C'=(A'\times B')\circ R$$

此处，$\mu_{(A'\times B')}(x,y)=\mu_{A'}(x)\wedge\mu_{B'}(y)$　或　$\mu_{(A'\times B')}(x,y)=\mu_{A'}(x)\,\mu_{B'}(y)$

3．模糊推理的性质

1）性质 1

若合成运算“$\circ$”采用最大–最小法或最大–积法，连接词“also”采用求并法，则“$\circ$”和“also”的运算次序可以交换，即

$$(A' \text{ and } B')\circ\bigcup_{i=1}^{n} R_i=\bigcup_{i=1}^{n}(A' \text{ and } B')\circ R_i$$

2）性质 2

若模糊蕴含关系采用 R_c 和 R_p 时，则有

$$C_i'=(A' \text{ and } B')\circ(A_i \text{ and } B_i\rightarrow C_i)=[A'\circ(A_i\rightarrow C_i)]\cap[B'\circ(B_i\rightarrow C_i)]$$

例 4-8　已知一个双输入单输出的模糊系统，其输入量为 x 和 y，输出量为 z，其输入/输出关系可用如下两条模糊规则描述：

R_1：如果 x 是 A_1 and y 是 B_1，则 z 是 C_1

R_2：如果 x 是 A_2 and y 是 B_2，则 z 是 C_2

现已知输入为 x 是 A' and y 是 B'，试求输出量 z。这里 x,y,z 均为模糊语言变量，且已知

$$A_1=\frac{1}{a_1}+\frac{0.5}{a_2}+\frac{0}{a_3},B_1=\frac{1}{b_1}+\frac{0.6}{b_2}+\frac{0.2}{b_3},C_1=\frac{1}{c_1}+\frac{0.4}{c_2}+\frac{0}{c_3}$$

$$A_2=\frac{0}{a_1}+\frac{0.5}{a_2}+\frac{1}{a_3},B_2=\frac{0.2}{b_1}+\frac{0.6}{b_2}+\frac{1}{b_3},C_2=\frac{0}{c_1}+\frac{0.4}{c_2}+\frac{1}{c_3}$$

$$A'=\frac{0.5}{a_1}+\frac{1}{a_2}+\frac{0.5}{a_3},B'=\frac{0.6}{b_1}+\frac{1}{b_2}+\frac{0.6}{b_3}$$

解：由于这里所有模糊集合的元素均为离散量，因此模糊集合可用模糊向量来描述，模糊关系可用模糊矩阵来描述。

（1）求每条规则的模糊蕴含关系 $R_i=(A_i \text{ and } B_i)\to C_i \quad (i=1,2)$

若此处 A_i and B_i 采用求交运算，蕴含关系采用最小运算 R_c，则

$$A_1 \quad \text{and} \quad B_1=A_1\times B_1=A_1^{\mathrm{T}}\circ B_1=[1 \quad 0.5 \quad 0]^{\mathrm{T}}\cdot[1 \quad 0.6 \quad 0.2]$$

$$=\begin{bmatrix}1\wedge1 & 1\wedge0.6 & 1\wedge0.2\\ 0.5\wedge1 & 0.5\wedge0.6 & 0.5\wedge0.2\\ 0\wedge1 & 0\wedge0.6 & 0\wedge0.2\end{bmatrix}=\begin{bmatrix}1 & 0.6 & 0.2\\ 0.5 & 0.5 & 0.2\\ 0 & 0 & 0\end{bmatrix}$$

为便于下面进一步的计算，可将 $A_1\times B_1$ 的模糊矩阵表示成如下的向量

$$\overline{R}_{A_1\times B_1}=[1 \quad 0.6 \quad 0.2 \quad 0.5 \quad 0.5 \quad 0.2 \quad 0 \quad 0 \quad 0]$$

则

$$R_1=(A_i \quad \text{and} \quad B_i)\to C_i=\overline{R}_{A_1\times B_1}^{\mathrm{T}}\wedge C_1=\begin{bmatrix}1\\0.6\\0.2\\0.5\\0.5\\0.2\\0\\0\\0\end{bmatrix}\wedge[1 \quad 0.4 \quad 0]=\begin{bmatrix}1 & 0.4 & 0\\0.6 & 0.4 & 0\\0.2 & 0.2 & 0\\0.5 & 0.4 & 0\\0.5 & 0.4 & 0\\0.2 & 0.2 & 0\\0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$$

同理可得

$$R_2=(A_2 \quad \text{and} \quad B_2)\to C_2=\overline{R}_{A_2\times B_2}^{\mathrm{T}}\wedge C_2=\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\\0 & 0.2 & 0.2\\0 & 0.4 & 0.5\\0 & 0.4 & 0.5\\0 & 0.2 & 0.2\\0 & 0.4 & 0.6\\0 & 0.4 & 1\end{bmatrix}$$

（2）求总的模糊蕴含关系 $\boldsymbol{R}$

$$\boldsymbol{R}=\boldsymbol{R}_1\cup\boldsymbol{R}_2=\begin{bmatrix}1 & 0.4 & 0\\0.6 & 0.4 & 0\\0.2 & 0.2 & 0\\0.5 & 0.4 & 0.2\\0.5 & 0.4 & 0.5\\0.2 & 0.4 & 0.5\\0 & 0.2 & 0.2\\0 & 0.4 & 0.6\\0 & 0.4 & 1\end{bmatrix}$$

（3）计算输入量的模糊集合

$$\boldsymbol{A}' \text{ and } \boldsymbol{B}'=\boldsymbol{A}'\times\boldsymbol{B}'=\boldsymbol{A}'^{\mathrm{T}}\circ\boldsymbol{B}'=\begin{bmatrix}0.5\\1\\0.5\end{bmatrix}\wedge\begin{bmatrix}0.6 & 1 & 0.6\end{bmatrix}=\begin{bmatrix}0.5 & 0.5 & 0.5\\0.6 & 1 & 0.6\\0.5 & 0.5 & 0.5\end{bmatrix}$$

$$\overline{\boldsymbol{R}}_{A\times B'}=\begin{bmatrix}0.5 & 0.5 & 0.5 & 0.6 & 1 & 0.6 & 0.5 & 0.5 & 0.5\end{bmatrix}$$

（4）计算输出量的模糊集合

$$\boldsymbol{C}'=(\boldsymbol{A}' \text{ and } \boldsymbol{B}')\circ\boldsymbol{R}=\overline{\boldsymbol{R}}_{A\times B'}\circ\boldsymbol{R}$$

$$=\begin{bmatrix}0.5 & 0.5 & 0.5 & 0.6 & 1 & 0.6 & 0.5 & 0.5 & 0.5\end{bmatrix}\circ\begin{bmatrix}1 & 0.4 & 0\\0.6 & 0.4 & 0\\0.2 & 0.2 & 0\\0.5 & 0.4 & 0.2\\0.5 & 0.4 & 0.5\\0.2 & 0.4 & 0.5\\0 & 0.2 & 0.2\\0 & 0.4 & 0.6\\0 & 0.4 & 1\end{bmatrix}=\begin{bmatrix}0.5 & 0.4 & 0.5\end{bmatrix}$$

最后求得输出量 z 的模糊集合为　$\boldsymbol{C}'=\dfrac{0.5}{c_1}+\dfrac{0.4}{c_2}+\dfrac{0.5}{c_3}$

4.2　模糊逻辑控制系统的基本结构

模糊控制是作为结合传统的并基于规则的专家系统、模糊集理论和控制理论的成果而诞生的，它与基于被控过程数学模型的传统控制理论有很大的区别。在模糊控制中，并不是

像传统控制那样需要对被控过程进行定量的数学建模，而是试图通过从能成功控制被控过程的领域专家那里获取知识，即专家行为和经验。当被控过程十分复杂甚至“病态”时，建立被控过程的数学模型或者不可能，或者需要高昂的代价，此时模糊控制就显得具有吸引力和使用性。由于人类专家的行为是实现模糊控制的基础，因此必须用一种容易且有效的方式来表达人类专家的知识。IF-THEN 规则格式是这种专家控制知识最合适的表示方式之一，即 IF “条件” THEN “结果”。这种表示方式有两个显著的特征：它们是定性的而不是定量的；它们是一种局部知识，这种知识将局部的“条件”与局部的“结果”联系起来。前者可用模糊子集表示，而后者需要模糊蕴含或模糊关系来表示。然而，当用计算机实现时，这种规则最终需具有数值形式。隶属函数和近似推理，为数值表示集合模糊蕴含提供了一种有利工具。

要实现一个实际的模糊控制系统，需要解决三个问题：知识的表示、推理策略和知识获取。知识表示是指如何将语言规则用数值方式表示出来；推理策略是指如何根据当前输入“条件”产生一个合理的“结果”；知识的获取解决如何获得一组恰当的规则。由于领域专家提供的知识常常是定性的，包含某种不确定性，因此，知识的表示和推理必须是模糊的或近似的，近似推理理论正是为满足这种需要而提出的。近似推理可看做是根据一些不精确的条件推导出一个精确结论的过程，许多学者对模糊表示、近似推理进行了大量的研究，在近似推理算法中，最广泛使用的是关系矩阵模型，它基于 L.A.Zadeh 的合成推理规则，首次由 Mamdani 采用。由于规则可被解释成逻辑意义上的蕴含关系，因此大量的蕴含算子已被提出并应用于实际中。

由此可见，模糊控制是以模糊集合论、模糊语言变量及模糊逻辑推理为基础的一种计算机控制。从线性控制与非线性控制的角度分类，模糊控制是一种非线性控制。从控制器智能性看，模糊控制属于智能控制的范畴，而且它已成为目前实现智能控制的一种重要而又有效的手段。尤其是模糊控制与神经网络、预测控制、遗传算法、混沌理论等新学科的结合，正在显示出其巨大的应用潜力。

4.2.1 模糊控制系统的组成

模糊控制系统由模糊控制器和控制对象组成，如图 4-7 所示。

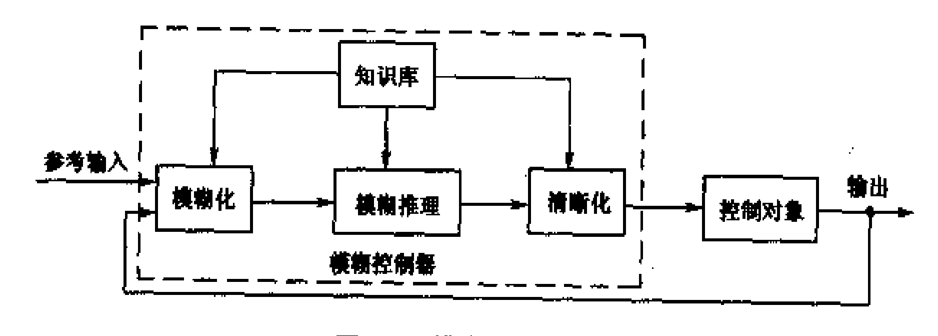


图 4-7 模糊控制系统的组成

4.2.2　模糊控制器的基本结构

模糊控制器的基本结构，如图 4-7 虚线框中所示。它主要包括以下四个部分。

1．模糊化

模糊化的作用是将输入的精确量转换成模糊化量。其输入量包括外界的参考输入、系统的输出或状态等。模糊化的具体过程如下：

（1）首先对这些输入量进行处理，以变成模糊控制器要求的输入量。例如，常见的情况是计算 $e=r-y$ 和 $\dot{e}=\mathrm{d}e/\mathrm{d}t$（式中，$r$ 表示参考输入；y 表示系统输出；e 表示误差）。有时为了减小噪声的影响，常常对 $\dot{e}$ 进行滤波后再使用，如可取 $\dot{e}=[s/(Ts+1)]e$；

（2）将上述已经处理过的输入量进行尺度变换，使其变换到各自的论域范围；

（3）将已经变换到论域范围的输入量进行模糊处理，使原先精确的输入量变成模糊量，并用相应的模糊集合来表示。

2．知识库

知识库中包含了具体应用领域中的知识和要求的控制目标。它通常由数据库和模糊控制规则库两部分组成。

（1）数据库主要包括各语言变量的隶属函数，尺度变换因子及模糊空间的分级数等。

（2）规则库包括了用模糊语言变量表示的一系列控制规则。它们反映了控制专家的经验和知识。

3．模糊推理

模糊推理是模糊控制器的核心，它具有模拟人的基于模糊概念的推理能力。该推理过程是基于模糊逻辑中的蕴含关系及推理规则来进行的。

4．清晰化

清晰化的作用是将模糊推理得到的控制量（模糊量）变换为实际用于控制的清晰量。它包含以下两部分内容：

（1）将模糊的控制量经清晰化变换，变成表示在论域范围的清晰量；

（2）将表示在论域范围的清晰量经尺度变换变成实际的控制量。

4.2.3　模糊控制器的维数

通常，将模糊控制器输入变量的个数称为模糊控制器的维数。下面以单输入单输出控制系统为例，给出几种结构形式的模糊控制器，如图 4-8 所示。

一般情况下，一维模糊控制器用于一阶被控对象，由于这种控制器输入变量只选一个误差，它的动态控制性能不佳，因此目前被广泛采用的均为二维模糊控制器，这种模糊控制器以误差和误差的变化为输入量，以控制量的变化为输出变量。从理论上讲，模糊控制器的

维数越高，控制就越精细。但是维数过高，模糊控制规则变得过于复杂，控制算法的实现就会相当困难。

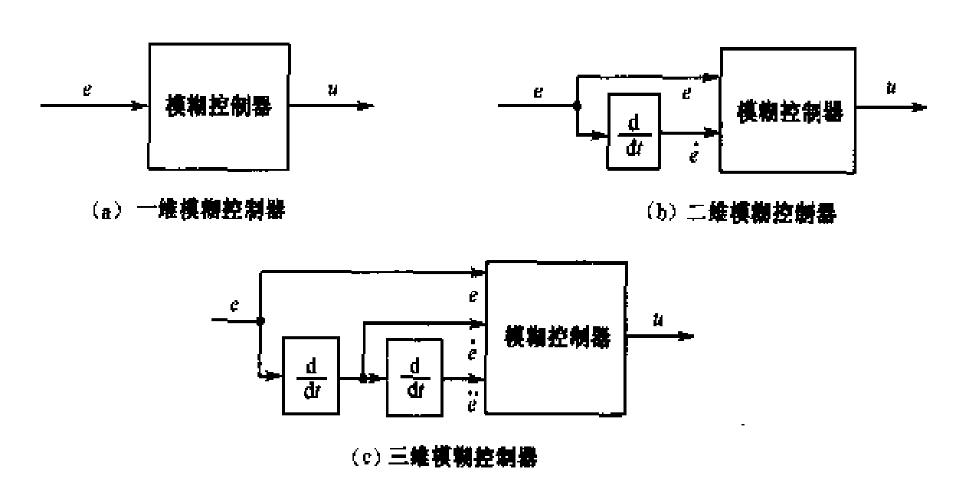


图 4-8　模糊控制器的结构

4.2.4　模糊控制中的几个基本运算操作

1. 模糊化运算

$$x=\mathrm{fz}(x_0)$$

式中，x_0 是输入的清晰量；x 是模糊集合；fz 表示模糊化运算符（fuzzifier）。

2. 句子连接运算

$$\boldsymbol{R}=\text{also}\ (\boldsymbol{R}_1, \boldsymbol{R}_2, \cdots, \boldsymbol{R}_n)$$

式中，$\boldsymbol{R}_i$ $(i=1, 2, \cdots, n)$是第 i 条规则所表示的模糊蕴含关系；$\boldsymbol{R}$ 是 n 个模糊关系的组合；组合运算用符号 also 表示。

3. 合成运算

$$z=(x \text{ and } y)\circ \boldsymbol{R}$$

式中，x 和 y 是输入模糊量；z 是输出模糊量；and 是句子连接运算符；“o” 是合成运算符。

4. 清晰化运算

以上推理过程得到的输出量 z 仍是模糊量，而实际的控制必须为清晰量。因此要进行如下的清晰化运算，即

$$z_0=\mathrm{df}(z)$$

式中，z 为控制输出的模糊量；z_0 为控制输出的清晰化量；df 表示清晰化运算符（defuzzifier）。

4.3　模糊逻辑控制系统的基本原理

4.3.1　模糊化运算

模糊化运算是将输入空间的观测量映射为输入论域上的模糊集合。模糊化在处理不确定信息方面具有重要的作用。在模糊控制中，观测到的数据常常是清晰量。由于模糊控制器对数据进行处理是基于模糊集合的方法，因此对输入数据进行模糊化是必不可少的一步。在进行模糊化运算之前，首先需要对输入量进行尺度变换，使其变换到相应的论域范围（后面将对此进行专门讨论）。下面所讨论的模糊化运算中的输入量均假定为已经过尺度变换的量。

在模糊控制中，主要采用以下两种模糊化方法。

1．单点模糊集合

若输入量数据 x_0 是准确的，则通常将其模糊化为单点模糊集合。设该模糊集合用 A 表示，则有

$$\mu_A(x)=\begin{cases}1 & (x=x_0)\\ 0 & (x\neq x_0)\end{cases}$$

其隶属度函数如图 4-9 所示。

这种模糊化方法只是形式上将清晰量转变成了模糊量，而实质上它表示的仍是准确量。在模糊控制中，当测量数据准确时，采用这样的模糊化方法是十分自然和合理的。

2．三角形模糊集合

若输入量数据存在随机测量噪声，则此时的模糊化运算相当于将随机量变换为模糊量。

对于这种情况，可以取模糊量的隶属度函数为等腰三角形，如图 4-10 所示。三角形的顶点对应于该随机数的均值，底边的长度等于 2σ。σ 表示该随机数据的标准差。隶属度函数取为三角形，主要是考虑其表示方便，计算简单。另一种常用的方法是取隶属度函数为铃形函数，即

$$\mu_A(x)=e^{-\frac{(x-x_0)^2}{2\sigma^2}}$$

它也就是正态分布的函数。

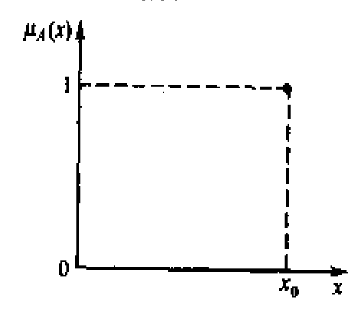


图 4-9　单点模糊集合的隶属度函数

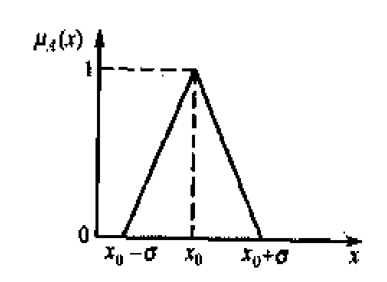


图 4-10　三角形模糊集合的隶属度函数

4.3.2 数据库

如前所述，模糊控制器中的知识库由两部分组成：数据库和模糊控制规则库。首先讨论数据库。数据库中包含了与模糊控制规则及模糊数据处理有关的各种参数，其中包括尺度变换参数、模糊空间分割和隶属度函数的选择等。

1. 输入量变换

对于实际的输入量，第一步首先需要进行尺度变换，将其变换到要求的论域范围。变换的方法可以是线性的，也可以是非线性的。例如，若实际的输入量为 x_0^*，其变化范围为$[x_{min}^*, x_{max}^*]$，要求的论域为$[x_{min}, x_{max}]$，且采用线性变换，则

$$x_0 = \frac{x_{min} + x_{max}}{2} + k\left(x_0^* - \frac{x_{max}^* + x_{min}^*}{2}\right)$$

$$k = \frac{x_{max} - x_{min}}{x_{max}^* - x_{min}^*}$$

式中，k 为比例因子。

论域可以是连续的，也可以是离散的。若要求离散的论域，则需要将连续的论域离散化或量化。量化可以是均匀的，也可以是非均匀的。表 4-1 和表 4-2 分别给出了均匀量化和非均匀量化的情况。

表 4-1 均匀量化

量化等级	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6
变化范围	≤ −5.5	(−5.5 −4.5]	(−4.5 −3.5]	(−3.5 −2.5]	(−2.5 −1.5]	(−1.5 −0.5]	(−0.5 0.5]	(0.5 1.5]	(1.5 2.5]	(2.5 3.5]	(3.5 4.5]	(4.5 5.5]	>5.5

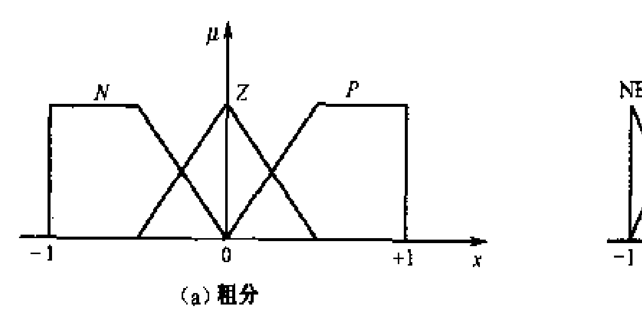
表 4-2 非均匀量化

量化等级	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6
变化范围	≤ −3.2	(−3.2 −1.6]	(−1.6 −0.8]	(−0.8 −0.4]	(−0.4 −0.2]	(−0.2 −0.1]	(−0.1 0.1]	(0.1 0.2]	(0.2 0.4]	(0.4 0.8]	(0.8 1.6]	(1.6 3.2]	>3.2

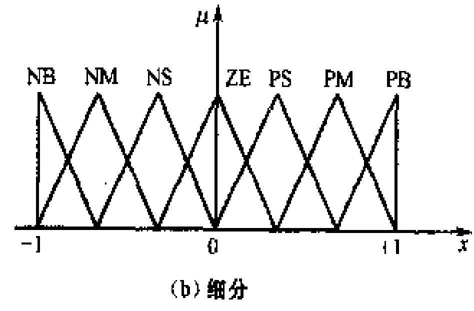
2. 输入和输出空间的模糊分割

模糊控制规则中的输入和前提的语言变量构成模糊输入空间，结论的语言变量构成模糊输出空间。每个语言变量的取值为一组模糊语言名称，它们构成了语言名称的集合。每个模糊语言名称对应一个模糊集合。对于每个语言变量，其取值的模糊集合具有相同的论域。模糊分割是要确定对于每个语言变量取值的模糊语言名称的个数，模糊分割的个数决定了模糊控制精细化的程度。这些语言名称通常均具有一定的含义。例如，NB：负大（Negative Big)；NM：负中（Negative Medium)；NS：负小（Negative Small)；ZE：零（Zero)；PS：正小（Positive Small)；PM：正中（Positive Medium)；PB：正大（Positive Big)。图 4-11

给出了两个模糊分割的例子，论域均为[−1, +1]，隶属度函数的形状为三角形或梯形。图 4-11（a）所示为模糊分割较粗的情况，图 4-11（b）所示为模糊分割较细的情况。图中所示的论域为正则化（normalization）的情况，即 $x\in[-1, +1]$，且模糊分割是完全对称的。这里假设尺度变换时已经做了预处理而变换成这样的标准情况。在一般情况下，模糊语言名称也可为非对称和非均匀地分布。



(a) 粗分



(b) 细分

图 4-11　两个模糊分割的图形表示

模糊分割的个数也决定了最大可能的模糊规则的个数，如对于两输入单输出的模糊系统，若 x 和 y 的模糊分割数分别为 3 和 7，则最大可能的规则数为 3×7=21。可见，模糊分割数越多，控制规则数也越多，因此模糊分割不可太细，否则需要确定太多的控制规则，这也是很困难的一件事。当然，模糊分割数太小将导致控制太粗略，难以对控制性能进行精心的调整。目前，尚没有一个确定模糊分割数的指导性的方法和步骤，它仍主要依靠经验和试凑。

3．完备性

对于任意的输入，模糊控制器均应给出合适的控制输出，这种性质称为完备性。模糊控制的完备性取决于数据库或规则库。

1）数据库方面

对于任意的输入，若能找到一个模糊集合，使该输入对于该模糊集合的隶属度函数不小于 ε，则称该模糊控制器满足 ε 完备性。图 4-11 所示即为 $\varepsilon = 0.5$ 的情况，它也是最常见的选择。

2）规则库方面

模糊控制的完备性对于规则库的要求是，对于任意的输入应确保至少有一个可适用的规则，而且规则的适用度应大于某个数，譬如说 0.5。根据完备性的要求，控制规则数不可太少。

4．模糊集合的隶属度函数

根据论域为离散和连续的不同情况，隶属度函数的描述也有如下两种方法。

1）数值描述方法

当论域为离散，且元素个数为有限时，模糊集合的隶属度函数可以用向量或者表格的形式来表示。表 4-3 给出了用表格表示的一个例子。

表 4-3　数值描述方法的隶属度

模糊集合 \ 隶属度 \ 元素	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6
NB	1.0	0.7	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
NM	0.3	0.7	1.0	0.7	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
NS	0.0	0.0	0.3	0.7	1.0	0.7	0.3	0.0	0.0	0.0	0.0	0.0	0.0
ZE	0.0	0.0	0.0	0.0	0.3	0.7	1.0	0.7	0.3	0.0	0.0	0.0	0.0
PS	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.7	1.0	0.7	0.3	0.0	0.0
PM	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.7	1.0	0.7	0.3
PB	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.7	1.0

在上面的表格中，每一行表示一个模糊集合的隶属度函数，如

$$\mathrm{NS}=\frac{0.3}{-4}+\frac{0.7}{-3}+\frac{1}{-2}+\frac{0.7}{-1}+\frac{0.3}{0}$$

2）函数描述方法

对于论域为连续的情况，隶属度常常用函数的形式来描述，最常见的有铃形函数、三角形函数、梯形函数等。下面给出铃形隶属度函数的解析式子

$$\mu_A(x)=\mathrm{e}^{-\frac{(x-x_0)^2}{2\sigma^2}}$$

式中，x_0 是隶属度函数的中心值；σ^2 是方差。图 4-12 为铃形隶属度函数的分布图。

图 4-12　铃形隶属度函数的分布图

隶属度函数的形状对模糊控制器的性能有很大影响。当隶属度函数比较窄瘦时，控制较灵敏；反之，控制较粗略和平稳。通常，当误差较小时，隶属度函数可取得较为窄瘦；误差较大时，隶属度函数可取得宽胖些。

4.3.3　规则库

模糊控制规则库由一系列“IF-THEN”型的模糊条件句构成。条件句的前件为输入变量，后件为控制变量。

1. 模糊控制规则的前件和后件变量的选择

模糊控制规则的前件和后件变量是指模糊控制器的输入和输出的语言变量。输出量即为控制量，它一般比较容易确定。输入量选什么以及选几个，则需要根据要求来确定。输入量比较常见的是误差 e 和它的导数 $\dot{e}$，有时还可以包括它的积分等。输入和输出语言变量的选择及其隶属度函数的确定，对于模糊控制器的性能有着十分关键的作用。它们的选择和确定主要依靠经验和工程知识。

2. 模糊控制规则的建立

模糊控制规则是模糊控制的核心。因此，如何建立模糊控制规则也就成了一个十分关键的问题。下面将讨论 4 种建立模糊控制规则的方法。它们之间并不是互相排斥的，相反，若能结合这几种方法，则可以更好地帮助建立模糊规则库。

1）基于专家的经验和控制工程知识

模糊控制规则具有模糊条件句的形式，它建立了前件中的输入变量与后件中的控制变量之间的联系。在日常生活中，用于决策的大部分信息主要是基于语义的方式而非数值的方式。因此，模糊控制规则是对人类行为和进行决策分析过程的最自然的描述方式。这也就是它为什么采用 IF-THEN 形式的模糊条件句的主要原因。

例如，电加热炉系统在阶跃输入 $y_r(t)$作用下，其输出 $y(t)$的过渡过程曲线，如图 4-13 所示。若借助专家对恒温控制的经验知识，则被调量 $y(t)$的调节过程大致如下：当 $y(t)$远小于 $y_r(t)$时，则大大增加控制量 $u(t)$；当 $y(t)$远大于 $y_r(t)$时，则大大减小控制量 $u(t)$；当 $y(t)$和 $y_r(t)$正负偏差不太大时，则根据 $y(t)$的变化趋势来确定控制量的大小。具体来说，当 $y(t)<y_r(t)$，被调量远离给定值（AB 段）时，增加控制量；当 $y(t)<y_r(t)$，被调量的变化有减小偏差的好趋势（BC 段）时，则综合考虑偏差大小和偏差变化率情况确定是稍增加、保持或减小控制量；当 $y(t)>y_r(t)$，被调量的变化有增加偏差的坏趋势（CD 段）时，则较多减少控制量；当 $y(t)>y_r(t)$，被调量变化平稳（DE 段）时，则减小控制量；当 $y(t)>y_r(t)$，被调量有减小偏差的好趋势（EF 段）时，则应综合考虑偏差大小和偏差变化率情况确定是减少、保持或稍增加控制量；当 $y(t)<y_r(t)$，被调量的变化有增加偏差的坏趋势（FG 段）时，则较大增加控制量。

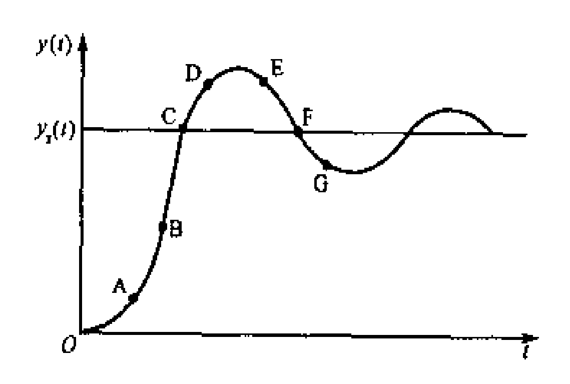


图 4-13　电加热炉系统单位阶跃响应曲线

基于上面的讨论，通过总结人类专家的经验，并用适当的语言来加以表述，最终可表示成模糊控制规则的形式。另一种方式是通过向有经验的专家和操作人员咨询，从而获得特定应用领域模糊控制规则的原型。在此基础上，再经一定的试凑和调整，可获得具有更好性

能的控制规则。

2）基于操作人员的实际控制过程

在许多人工控制的工业系统中，很难建立控制对象的模型，因此，用常规的控制方法来对其进行设计和仿真比较困难。而熟练的操作人员却能成功地控制这样的系统。事实上，操作人员有意或无意地使用了一组 IF-THEN 模糊规则来进行控制。但是，他们往往并不能用语言明确地将它们表达出来，因此，可以通过记录操作人员实际控制过程时的输入输出数据，并从中总结出模糊控制规则。

3）基于过程的模糊模型

控制对象的动态特性通常可用微分方程、传递函数、状态方程等数学方法来加以描述，这样的模型称为定量模型或清晰化模型。控制对象的动态特性也可以用语言的方法来描述，这样的模型称为定性模型或模糊模型。基于模糊模型，也能建立起相应的模糊控制规律。这样设计的系统是纯粹的模糊系统，即控制器和控制对象均是用模糊的方法来加以描述的，因而它比较适合于采用理论的方法来进行分析和控制。

4）基于学习

许多模糊控制主要是用来模仿人的决策行为，但很少具有类似人的学习功能，即根据经验和知识产生模糊控制规则并对它们进行修改的能力。Mamdani 于 1979 年首先提出了模糊自组织控制，它便是一种具有学习功能的模糊控制。该自组织控制具有分层递阶的结构，它包含有两个规则库：第一个规则库是一般的模糊控制的规则库；第二个规则库由宏规则组成，它能够根据对系统的整体性能要求来产生并修改一般的模糊控制规则，从而显示了类似人的学习能力。自 Mamdani 的工作之后，近来又有不少人在这方面做了大量的研究工作。最典型的例子是 Sugeno 的模糊小车。它是具有学习功能的模糊控制车，经过训练后能够自动地停靠在要求的位置。

3. 模糊控制规则的类型

在模糊控制中，目前主要应用如下两种形式的模糊控制规则。

1）状态评估模糊控制规则

它具有如下的形式：

$\boldsymbol{R}_1$：如果 x 是 A_1 and y 是 $\boldsymbol{B}_1$，则 z 是 $\boldsymbol{C}_1$

also $\boldsymbol{R}_2$：如果 x 是 A_2 and y 是 $\boldsymbol{B}_2$，则 z 是 $\boldsymbol{C}_2$

…

also $\boldsymbol{R}_n$：如果 x 是 A_n and y 是 $\boldsymbol{B}_n$，则 z 是 $\boldsymbol{C}_n$。

在现有的模糊控制系统中，大多数情况均采用这种形式。前面所讨论的也都是这种情形。

对于更一般的情形，模糊控制规则的后件可以是过程状态变量的函数，即

$\boldsymbol{R}_i$：如果 x 是 A_i ⋯ and y 是 $\boldsymbol{B}_i$，则 $z=f_i(x, \cdots, y)$

它根据对系统状态的评估，按照一定的函数关系计算出控制作用 z。

2）目标评估模糊控制规则

典型的形式如下：

$\boldsymbol{R}_i$：如果[u 是 C_i→(x 是 A_i and y 是 B_i)]，则 u 是 C_i

式中，u 是系统的控制量；x 和 y 表示要求的状态和目标或者是对系统性能的评估，因而 x 和 y 的取值常常是“好”、“差”等模糊语言。对于每个控制命令 C_i，可通过预测相应的结果（x, y），从中选用最适合的控制规则。

上面的规则可进一步解释为：当控制命令选 C_i 时，若性能指标 x 是 A_i, y 是 B_i，则选用该条规则，并将 C_i 取为控制器的输出。例如，在日本仙台的地铁模糊自动火车运行系统中，就采用了这种类型的模糊控制规则。列出其中典型的一条，如“如果控制标志不改变，则火车停在预定的容许区域，那么控制标志不改变”。

采用目标评估模糊控制规则，对控制的结果加以预测，并根据预测的结果来确定采取的控制行动。它本质上是一种模糊预测控制。

4. 模糊控制规则的其他性能要求

1）模糊控制规则数

若模糊控制器的输入有 m 个，每个输入的模糊分级数分别为 $n_1, n_2, \cdots, n_m$，则最大可能的模糊规则数为 $N_{\max}=n_1n_2\cdots n_m$，实际的模糊控制数应该取多少取决于很多因素，目前尚无普遍适用的一般步骤。总的原则是，在满足完备性的条件下，尽量取较少的规则数，以简化模糊控制器的设计和便于实现。

2）模糊控制规则的一致性

模糊控制规则主要基于操作人员的经验，它取决于对多种性能的要求，而不同的性能指标要求往往互相制约，甚至是互相矛盾的。这就要求模糊控制不能出现互相矛盾的情况。

4.3.4 模糊推理

模糊控制中的规则通常来源于专家的知识，在模糊控制中，通过用一组语言描述的规则来表示专家的知识，通常具有如下形式，即

IF（满足一组条件）THEN（可以推出一组结论）

在 IF-THEN 规则中的输入和前提条件及结论均是模糊的概念，如“若温度偏高，则加入较多的冷却水”，其中“偏高”和“较多”均为模糊量。常常称这样的 IF-THEN 规则为模糊条件句。因此，在模糊控制中，模糊控制规则也就是模糊条件句。其中前提条件为具体应用领域中的条件，结论为要采取的控制行动。IF-THEN 的模糊控制规则为表示控制领域的专家知识提供了方便的工具。对于多输入多输出（MIMO）模糊系统，则有多个输入和前提条件以及多个结论。

对于多输入多输出模糊控制器，其规则库具有如下形式：

$$R=\{R^1_{\text{MIMO}},\ R^2_{\text{MIMO}},\ \cdots,\ R^n_{\text{MIMO}}\}$$

式中，R^i_{MIMO}：如果 x 是 A_i and ⋯ and y 是 B_i，则 z_1 是 C_{i1}，⋯，z_q 是 C_{iq}。

R^i_{MIMO} 的前件（输入和前提条件）是直积空间 $X\times\cdots\times Y$ 上的模糊集合，后件（结论）是 q 个控制作用的并，它们之间是互相独立的。因此，第 i 条规则 R^i_{MIMO} 可以表示为如下的模糊蕴含关系，即

$$R^i_{\text{MIMO}}:\ (A_i\times\cdots\times B_i)\to(C_{i1}+\cdots+C_{iq})$$

于是规则 R^i_{MIMO} 可以表示为

$$\begin{aligned}R^i_{\text{MIMO}}&=\{(A_i\times\cdots\times B_i)\to(C_{i1}+\cdots+C_{iq})\}\\&=\{[(A_i\times\cdots\times B_i)\to C_{i1}],\cdots,[(A_i\times\cdots\times B_i)\to C_{iq}]\}\\&=\{R^{i1}_{\text{MISO}},R^{i2}_{\text{MISO}},\cdots,R^{iq}_{\text{MISO}}\}\end{aligned}$$

规则库 R 可以表示为

$$\begin{aligned}R&=\{\bigcup_{i=1}^{n}R^i_{\text{MIMO}}\}=\{\bigcup_{i=1}^{n}[(A_i\times\cdots\times B_i)\to(C_{i1}+\cdots+C_{iq})]\}\\&=\{\bigcup_{i=1}^{n}[(A_i\times\cdots\times B_i)\to C_{i1}],\cdots,\bigcup_{i=1}^{n}[(A_i\times\cdots\times B_i)\to C_{iq}]\}\\&=\{R^1_{\text{MISO}},R^2_{\text{MISO}},\cdots,R^q_{\text{MISO}}\}\end{aligned}$$

可见，规则库 R 可看成由 q 个子规则库所组成，每一个子规则库由 n 个多输入单输出的规则所组成。由于各个子规则是互相独立的，因此下面只考虑 MIMO 中一个子规则库的模糊推理问题，即只需考虑 MISO 子系统的模糊推理问题。其中第 i 条规则 R^i_{MIMO} 是由 q 个独立的 MISO 规则组成的，即

$$R^i_{\text{MIMO}}=\{R^{i1}_{\text{MISO}},\ R^{i2}_{\text{MISO}},\ \cdots,\ R^{iq}_{\text{MISO}}\}$$

式中，R^{ij}_{MISO}：如果 x 是 A_i and⋯and y 是 B_i，则 z_j 是 C_{ij}。

为了不失一般性，考虑两个输入一个输出的模糊控制器。设已建立的模糊控制规则库为

R_1：如果 x 是 A_1 and y 是 B_1，则 z 是 C_1

also R_2：如果 x 是 A_2 and y 是 B_2，则 z 是 C_2

⋯

also R_n：如果 x 是 A_n and y 是 B_n，则 z 是 C_n。

式中，x, y 和 z 是代表系统状态和控制量的语言变量，x 和 y 为输入量，z 为控制量；A_i, B_i 和 C_i（i=1, 2, ⋯, n）分别是语言变量 x, y, z 在其论域 X, Y, Z 上的语言变量值，所有规则组合在一起构成了规则库。

对于第 i 条规则“如果 x 是 A_i and y 是 B_i，则 z 是 C_i”的模糊蕴含关系 R_i 定义为

$$R_i=(A_i\ \text{and}\ B_i)\to C_i$$

即

$$\mu_{R_i}=\mu_{(A_i\text{and}B_i)\to C_i}(x,y,z)=[\mu_{A_i}(x)\text{and}\mu_{B_i}(y)]\to\mu_{C_i}(z)$$

式中，“A_i and B_i” 是定义在 $X \times Y$ 上的模糊集合 $A_i \times B_i$，$R_i = (A_i \text{ and } B_i) \rightarrow C_i$ 是定义在 $X \times Y \times Z$ 上的模糊蕴含关系。

所有 n 条模糊控制规则的总模糊蕴含关系为（取连接词“also”为求并运算）

$$R = \bigcup_{i=1}^{n} R_i$$

设已知模糊控制器的输入模糊量为：x 是 A' and y 是 B'，则根据模糊控制规则进行模糊推理，可以得出输出模糊量 z（用模糊集合 C' 表示）为

$$C' = (A' \text{ and } B') \circ R$$

其中 $\mu_{(A' \text{and} B')}(x, y) = \mu_{A'}(x) \wedge \mu_{B'}(y)$ 或 $\mu_{(A' \text{and} B')}(x, y) = \mu_{A'}(x)\, \mu_{B'}(y)$

以上运算包括了三种主要的模糊逻辑运算：and 运算、合成运算“∘”和蕴含运算“→”。在模糊控制中，and 运算通常采用求交（取小）或求积（代数积）的方法；合成运算“∘”采用最大–最小或最大–积（代数积）的方法；蕴含运算“→”采用求交（R_c）或求积（R_p）的方法。

4.3.5 清晰化计算

以上通过模糊推理得到的是模糊量，而对于实际的控制则必须为清晰量，因此需要将模糊量转换成清晰量，这就是清晰化计算所要完成的任务。清晰化计算通常有以下几种方法。

1. 平均最大隶属度法（mom）

若输出量模糊集合 C' 的隶属度函数只有一个峰值，则取隶属度函数的最大值为清晰值，即

$$\mu_{C'}(z_0) \geqslant \mu_{C'}(z) \quad z \in Z$$

式中，z_0 表示清晰值。若输出量的隶属度函数有多个极值，则取这些极值的平均值为清晰值。

例 4-9 已知输出量 z_1 模糊集合为

$$C_1' = \frac{0.1}{2} + \frac{0.4}{3} + \frac{0.7}{4} + \frac{1.0}{5} + \frac{0.7}{6} + \frac{0.3}{7}$$

z_2 的模糊集合为

$$C_2' = \frac{0.3}{-4} + \frac{0.8}{-3} + \frac{1}{-2} + \frac{1}{-1} + \frac{0.8}{0} + \frac{0.3}{1} + \frac{0.1}{2}$$

求相应的清晰量 z_{10} 和 z_{20}。

根据最大隶属度法，很容易求得

$$z_{10} = \text{df}(z_1) = 5$$

$$z_{20} = \text{df}(z_2) = (-2-1)/2 = -1.5$$

2．最大隶属度取最小值方法（som）

该方法取模糊集合中具有最大隶属度的所有点中的最小的一个作为去模糊化的结果。

3．最大隶属度取最大值方法（lom）

该方法取模糊集合中具有最大隶属度的所有点中的最大的一个作为去模糊化的结果。

4．中位数法（面积平分法 bisector）

中位数法是取$\mu_C(z)$的中位数作为 z 的清晰量，即 z_0=df(z)= $\mu_C(z)$的中位数，它满足

$$\int_a^{z_0}\mu_{C'}(z)\mathrm{d}z=\int_{z_0}^{b}\mu_{C'}(z)\mathrm{d}z$$

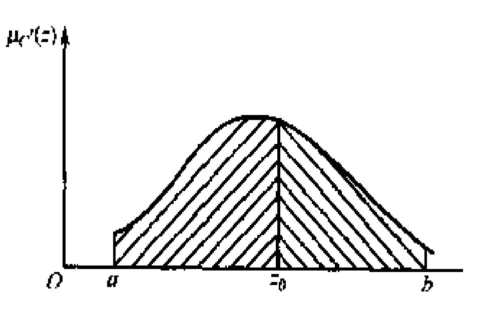


图 4-14　清晰化计算的中位数法

也就是说，以 z_0 为分界，a 为下界，b 为上界，$\mu_C(z)$与 z 轴之间面积两边相等，如图 4-14 所示。

5．加权平均法（面积重心法 centroid）

加权平均法取$\mu_C(z)$的加权平均值为 z 的清晰值，即

$$z_0=\mathrm{df}(z)=\frac{\int_a^b z\mu_{C'}(z)\mathrm{d}z}{\int_a^b \mu_{C'}(z)\mathrm{d}z}$$

它类似于重心的计算，所以也称重心法。对于论域为离散的情况，则有

$$z_0\mathrm{df}=\frac{\sum_{i=1}^{n} z_i\mu_{C'}(z_i)}{\sum_{i=1}^{n}\mu_{C'}(z_i)}$$

例 4-10　题设条件同例 4-9，用加权平均法计算清晰值 z_{10} 和 z_{20}。

$$z_{10}=\frac{0.1\times2+0.4\times3+0.7\times4+1\times5+0.7\times6+0.3\times7}{0.1+0.4+0.7+1+0.7+0.3}=4.84$$

$$z_{20}=\frac{0.3\times(-4)+0.8\times(-3)+1\times(-2)+1\times(-1)+0.8\times0+0.3\times1+0.1\times2}{0.3+0.8+1+1+0.8+0.3+0.1}=-1.42$$

在以上各种清晰化方法中，加权平均法应用最为普遍。

在求得清晰值 z_0 后，还需经尺度变换变为实际的控制量。变换的方法可以是线性的，也可以是非线性的。若 z_0 的变化范围为［z_{min}, z_{max}］，实际控制量的变化范围为［u_{min}, u_{max}］，采用线性变换，则

$$u=\frac{u_{\max}+u_{\min}}{2}+k\left(z_0-\frac{z_{\max}+z_{\min}}{2}\right)$$

$$k=\frac{u_{\max}-u_{\min}}{z_{\max}-z_{\min}}$$

式中，k 为比例因子。

4.4 离散论域的模糊控制系统的设计

当论域为离散时，经过量化后的输入量的个数是有限的。因此，可以针对输入情况的不同组合，离线计算出相应的控制量，从而组成一张控制表，实际控制时只要直接查这张控制表即可，在线的运算量是很少的。这种离线计算、在线查表的模糊控制方法比较容易满足实时控制的要求。图 4-15 所示为论域为离散时的模糊控制系统的结构，图中假设采用误差 e 和误差导数 $\dot{e}$ 作为模糊控制器的输入量，这是最常使用的情况。

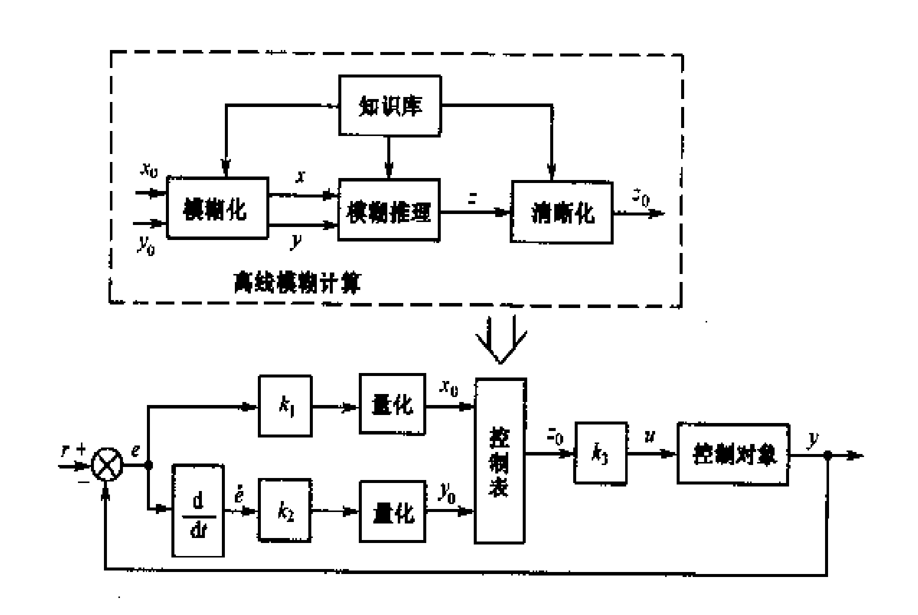


图 4-15 论域为离散时的模糊控制系统结构

图 4-15 中，k_1、k_2 和 k_3 为尺度变换的比例因子。设 e、$\dot{e}$ 和 u 的实际变化范围分别为 $[-e_m,e_m]$、$[-\dot{e}_m,\dot{e}_m]$ 和 $[-u_m,u_m]$，并设 x、y 和 z 的论域分别为

$$\{-n_i,\cdots,-1,0,1,\cdots,n_i\} \quad (i=1,2,3)$$

则

$$k_1=n_1/e_m,\ k_2=n_2/\dot{e}_m,\ k_3=u_3/n_3 \tag{4-1}$$

图 4-15 中，量化的功能是将比例变换后的连续值经四舍五入变为整数量。

从 x_0、y_0 到 z_0 的模糊推理计算过程采用前面已经讨论过的方法进行。由于 x_0、y_0 的个数是有限的，因此可以将它们的所有可能的组合情况先计算出来（即图中的离线模糊计算部

分)，将计算的结果列成一张控制表。实际控制时只需查询该控制表即可由 x_0、y_0 求得 z_0。求得 z_0 后再经比例变换，变成实际的控制量。

在该例中，控制器的输入量为 e 和 $\dot{e}$，因此它相当于是非线性的 PD 控制，k_1、k_2 分别是比例项和导数项前面的比例系数，它们对系统性能有很大影响，要仔细地加以选择。k_3 串联在系统的回路中，它直接影响整个回路的增益，因此 k_3 也对系统的性能有很大影响，一般说来，k_3 选得大，系统反应快。但过大有可能使系统不稳定。

下面通过一个具体例子来说明离线模糊计算的过程。

设

$$X,Y,Z \in \{-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6\}$$

$T(x)$={NB(负大), NM(负中), NS(负小), NZ(负零), PZ(正零), PS(正小), PM(正中), PB(正大)}

$T(y)$=$T(z)$={NB, NM, NS, ZE(零), PS, PM, PB}

表 4-4 给出了语言变量 x 的隶属度函数。y 和 z 的隶属度函数同表 4-3。

表 4-4　语言变量 x 的隶属度函数

模糊集合 \ 隶属度 \ x	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6
NB	1.0	0.8	0.7	0.4	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
NM	0.2	0.7	1.0	0.7	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
NS	0.0	0.1	0.3	0.7	1.0	0.7	0.2	0.0	0.0	0.0	0.0	0.0	0.0
NZ	0.0	0.0	0.0	0.0	0.1	0.6	1.0	0.0	0.0	0.0	0.0	0.0	0.0
PZ	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.6	0.1	0.0	0.0	0.0	0.0
PS	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.7	1.0	0.7	0.3	0.1	0.0
PM	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.7	1.0	0.7	0.3
PB	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.4	0.7	0.8	1.0

表 4-4 和表 4-3 是一种表示离散论域的模糊集合及其隶属度函数的简捷形式。例如，对于表 4-4，它表示

$$\mathrm{NB}=\frac{1.0}{-6}+\frac{0.8}{-5}+\frac{0.7}{-4}+\frac{0.4}{-3}+\frac{0.1}{-2},\mathrm{NM}=\frac{0.2}{-6}+\frac{0.7}{-5}+\frac{1.0}{-4}+\frac{0.7}{-3}+\frac{0.3}{-2}+\cdots,$$

$$\mathrm{PB}=\frac{0.1}{2}+\frac{0.4}{3}+\frac{0.7}{4}+\frac{0.8}{5}+\frac{1.0}{6}$$

表 4-5 列出了该模糊控制器所采用的模糊控制规则。

表 4-5　模糊控制规则

x \ z \ y	NB	NM	NS	ZE	PS	PM	PB
NB	NB	NB	NB	NB	NM	ZE	ZE
NM	NB	NB	NB	NB	NM	ZE	ZE
NS	NM	NM	NM	NM	ZE	PS	PS
NZ	NM	NM	NS	ZE	PS	PM	PM
PZ	NM	NM	NS	ZE	PS	PM	PM
PS	NS	NS	ZE	PM	PM	PM	PM
PM	ZE	ZE	PM	PB	PB	PB	PB
PB	ZE	ZE	PM	PB	PB	PB	PB

表 4-5 是表示模糊控制规则的简捷形式。该表中共包含 56 条规则，由于 x 的模糊分割数为 8，y 的模糊分割数为 7，因此该表包含了最大可能的规则数。一般情况下，规则数可以少于 56，这时表中相应栏内可以为空。表 4-5 中所表示的规则依次为

$\boldsymbol{R}_1$：如果 x 是 NB and y 是 NB，则 z 是 NB

$\boldsymbol{R}_2$：如果 x 是 NB and y 是 NM，则 z 是 NB

…

$\boldsymbol{R}_{56}$：如果 x 是 PB and y 是 PB，则 z 是 PB

设已知输入为 x_0 和 y_0，模糊化运算采用单点模糊集合，则相应的输入量模糊集合 $\boldsymbol{A}'$ 和 $\boldsymbol{B}'$分别为

$$\mu_{A'}(x)=\begin{cases}1 & (x=x_0)\\ 0 & (x\neq x_0)\end{cases},\mu_{B'}(y)=\begin{cases}1 & (y=y_0)\\ 0 & (y\neq y_0)\end{cases}$$

根据前面介绍的模糊推理方法及性质，可求得输出量的模糊集合 $\boldsymbol{C}'$为（假设 and 用求交法，also 用求并法，合成用最大–最小法，模糊蕴含用求交法）

$$\boldsymbol{C}'=(\boldsymbol{A}'\times\boldsymbol{B}')\circ\boldsymbol{R}=(\boldsymbol{A}'\times\boldsymbol{B}')\circ\bigcup_{i=1}^{56}\boldsymbol{R}_i=\bigcup_{i=1}^{56}(\boldsymbol{A}'\times\boldsymbol{B}')\circ[(\boldsymbol{A}_i\times\boldsymbol{B}_i)\to\boldsymbol{C}_i]$$

$$=\bigcup_{i=1}^{56}[\boldsymbol{A}'\circ(\boldsymbol{A}_i\to\boldsymbol{C}_i)]\bigcap[\boldsymbol{B}'\circ(\boldsymbol{B}_i\to\boldsymbol{C}_i)]=\bigcup_{i=1}^{56}\boldsymbol{C}'_{iA}\bigcap\boldsymbol{C}'_{iB}=\bigcup_{i=1}^{56}\boldsymbol{C}'_i$$

下面以 $x_0=-6$，$y_0=-6$ 为例说明计算过程。此时有

$$\boldsymbol{A}'=[1\ \ 0\ \ \cdots\ \ 0]_{1\times13},\boldsymbol{B}'=[1\ \ 0\ \ \cdots\ \ 0]_{1\times13}$$

对于表 4-5 第 1 行第 1 列的规则：若 x 为 NB and y 为 NB，则 z 为 NB

根据表 4-4 和表 4-3 可得

$$\boldsymbol{A}_{\mathrm{NB}}=[1\ \ 0.8\ \ 0.7\ \ 0.4\ \ 0.1\ \ 0\ \ \cdots\ \ 0]_{1\times13},\boldsymbol{B}_{\mathrm{NB}}=[1\ \ 0.7\ \ 0.3\ \ 0\ \ \cdots\ \ 0]_{1\times13}$$

$$\boldsymbol{C}_{\mathrm{NB}}=[1\ \ 0.7\ \ 0.3\ \ 0\ \ \cdots\ \ 0]_{1\times13},$$

$$R_{1A}=A_1\to C_1=A_{\mathrm{NB}}\to C_{\mathrm{NB}}=\begin{bmatrix}1\\0.8\\0.7\\0.4\\0.1\\0\\\vdots\\0\end{bmatrix}\wedge[1\ \ 0.7\ \ 0.3\ \ 0\ \ \cdots\ \ 0]=\begin{bmatrix}1&0.7&0.3&\\0.8&0.7&0.3&\\0.7&0.7&0.3&\mathbf{0}\\0.4&0.4&0.3&\\0.1&0.1&0.1&\\&\mathbf{0}&&\mathbf{0}\end{bmatrix}_{13\times13}$$

$$C'_{1A}=A'\circ(A_1\to C_1)=[1\ \ 0\ \ \cdots\ \ 0]_{1\times13}\circ R_{1A}=[1\ \ 0.7\ \ 0.3\ \ 0\ \ \cdots\ \ 0]_{1\times13}$$

$$R_{1B}=B_1\to C_1=B_{\mathrm{NB}}\to C_{\mathrm{NB}}=\begin{bmatrix}1\\0.7\\0.3\\0\\\vdots\\0\end{bmatrix}\wedge[1\ \ 0.7\ \ 0.3\ \ 0\ \ \cdots\ \ 0]=\begin{bmatrix}1&0.7&0.3&\\0.7&0.7&0.3&\\0.3&0.3&0.3&\mathbf{0}\\0&0&0&\\0&0&0&\\&\mathbf{0}&&\mathbf{0}\end{bmatrix}_{13\times13}$$

$$C'_{1B}=B'\circ(B_1\to C_1)=[1\ \ 0\ \ \cdots\ \ 0]_{1\times13}\circ R_{1B}=[1\ \ 0.7\ \ 0.3\ \ 0\ \ \cdots\ \ 0]_{1\times13}$$

$$C'_1=C'_{1A}\cap C'_{1B}=[1\ \ 0.7\ \ 0.3\ \ 0\ \ \cdots\ \ 0]_{1\times13}$$

对于表 4-5 第 1 行第 2 列的规则：若 x 为 NB and y 为 NM，则 z 为 NB

根据表 4-4 和表 4-3 可得

$$A_{\mathrm{NB}}=[1\ \ 0.8\ \ 0.7\ \ 0.4\ \ 0.1\ \ 0\ \ \cdots\ \ 0]_{1\times13}$$
$$B_{\mathrm{NM}}=[0.3\ \ 0.7\ \ 1\ \ 0.7\ \ 0.3\ \ 0\ \ \cdots\ \ 0]_{1\times13}$$
$$C_{\mathrm{NB}}=[1\ \ 0.7\ \ 0.3\ \ 0\ \ \cdots\ \ 0]_{1\times13}$$

$$R_{2A}=A_2\to C_2=\mathrm{A}_{\mathrm{NB}}\to C_{\mathrm{NB}}=R_{1A}$$
$$C'_{2A}=A'\circ(A_2\to C_2)=A'\circ(A_{\mathrm{NB}}\to C_{\mathrm{NB}})=C'_{1A}=[1\ \ 0.7\ \ 0.3\ \ 0\ \ \cdots\ \ 0]_{1\times13}$$

$$R_{2B}=B_2\to C_2=B_{\mathrm{NM}}\to C_{\mathrm{NB}}=\begin{bmatrix}0.3\\0.7\\1\\0.7\\0.3\\0\\\vdots\\0\end{bmatrix}\wedge[1\ \ 0.7\ \ 0.3\ \ 0\ \ \cdots\ \ 0]=\begin{bmatrix}0.3&0.3&0.3&\\0.7&0.7&0.3&\\1&0.7&0.3&\mathbf{0}\\0.7&0.7&0.3&\\0.3&0.3&0.3&\\&\mathbf{0}&&\mathbf{0}\end{bmatrix}_{13\times13}$$

$$C'_{2B}=B'\circ(B_2\to C_2)=[1\ \ 0\ \ \cdots\ \ 0]_{1\times13}\circ R_{2B}=[0.3\ \ 0.3\ \ 0.3\ \ 0\ \ \cdots\ \ 0]_{1\times13}$$

$$C'_2 = C'_{2A} \cap C'_{2B} = \begin{bmatrix} 0.3 & 0.3 & 0.3 & 0 & \cdots & 0 \end{bmatrix}_{1\times 13}$$

按同样的方法依次求出 $C'_3, C'_4 \cdots, C'_{56}$，最终求得

$$C' = \bigcup_{i=1}^{56} C'_i = \begin{bmatrix} 1 & 0.7 & 0.3 & 0 & \cdots & 0 \end{bmatrix}_{1\times 13}$$

对所求得的输出量模糊集合进行清晰化计算（用加权平均法）得

$$z'_0 = \mathrm{df}(z) = \frac{1\times(-6)+0.7\times(-5)+0.3\times(-4)}{1+0.7+0.3} = -5.35$$

按照同样的步骤，可以计算出当 x_0, y_0 为其他组合时的输出量 z_0。最后可列出实际查询的控制表 4-6。

表 4-6　控制表

x_0 \ z_0 \ y_0	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6
−6	−5.35												0
−5													
−4			−4.69								−0.69		
−3													
−2					−3.47				0.44				
−1													
0							0						
1													
2					−0.44				3.47				
3													
4			0.69								4.69		
5													
6	0												5.35

对于如图 4-15 所示的模糊控制系统，可以利用 Simulink 对其进行仿真，仿真框图如图 4-16 所示。

将图 4-16 中两个 MATLAB Function 模块的 MATLAB Function 对话框中均改为四舍五入 round 函数；二维表格 Look-Up Table(2-D)中的值根据表 4-6 的内容来填写；传递函数(Transfer Fcn)根据被控系统的模型进行设置；放大器的参数 k_1、k_2、k_3 的值根据式（4-1）的计算结果进行设置。

量化因子 k_1、k_2 的大小对控制系统的动态性能影响很大。k_1 选择较大时，系统的超调较大，过渡过程较长。这一点也不难理解，因为从理论上讲，k_1 增大，相当于缩小了误差的基

本论域，增大了误差变量的控制作用，因此导致上升时间变短，但由于出现超调，使得系统的过渡过程变长。k_2 选择较大时，超调量减小，k_2 选择越大，超调量越小，但系统的响应速度变慢。k_2 对超调的遏制作用十分明显。

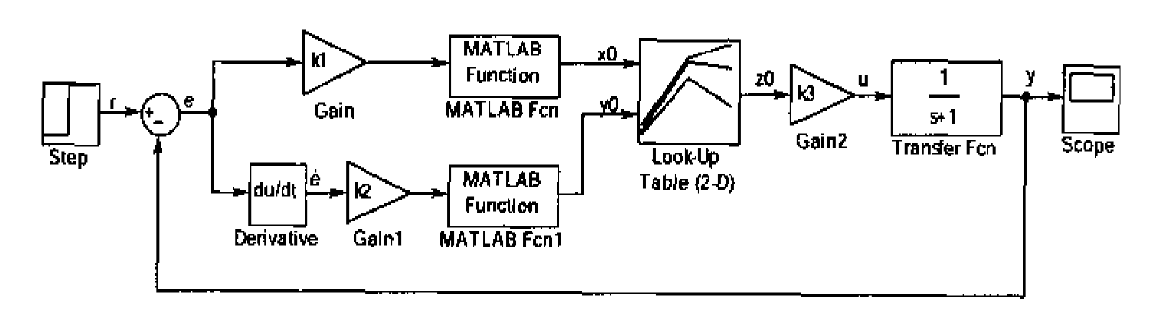


图 4-16 Simulink 系统仿真框图

量化因子 k_1、k_2 的大小意味着对输入变量误差和误差变化的不同加权程度，k_1、k_2 二者之间也相互影响，在选择量化因子 k_1、k_2 时要充分考虑到这一点。

输出比例因子 k_3 的大小也影响着模糊控制系统的特性。k_3 选择过小，会使系统动态响应过程变长，而 k_3 选择过大，会导致系统振荡。输出比例因子 k_3 作为模糊控制器的总的增益，它的大小影响着控制器的输出，通过调整 k_3 可以改变对被控对象输入的大小。

应该指出，量化因子和比例因子的选择并不是惟一的，可能有几组不同的值，都能使系统获得较好的响应特性。对于比较复杂的被控过程，有时采用一组固定的量化因子和比例因子难以收到预期的控制效果，可以在控制过程中采用改变量化因子和比例因子的方法，来调整整个控制过程中的不同阶段上的控制特性，以使对复杂过程控制得到满意的控制效果。这种形式的控制器称为自调整比例因子模糊控制器。

4.5 具有 PID 功能的模糊控制器

在常规控制中，PID 控制是最简单实用的一种控制方法，它既可以依靠数学模型通过解析的方法进行设计，也可不依赖模型而凭借经验和试凑来确定。前面讨论的模糊控制，一般均假设用误差 e 和误差导数 $\dot{e}$ 作为模糊控制的输入量，因而它本质上相当于一种非线性 PD 控制。为了消除稳态误差，需要加入积分作用，图 4-17 给出了两种典型的具有 PID 功能的模糊控制器的结构图，简称为模糊 PID 控制器，其中图 4-17（a）为常规模糊 PID 控制，图 4-17（b）为增量模糊 PID 控制。

在如图 4-17 所示的典型结构中，模糊控制器有三个输入。若每个输入量分为 7 个等级，则最多可能需要 7^3=343 条模糊规则；而当输入量为两个时，最多只需要 7^2=49 条模糊规则。可见，增加了一个输入量，大大增加了模糊控制器设计和计算的复杂性。为此，可以考虑采用如图 4-18 所示的变形结构，它同样可以实现模糊 PID 控制的功能。

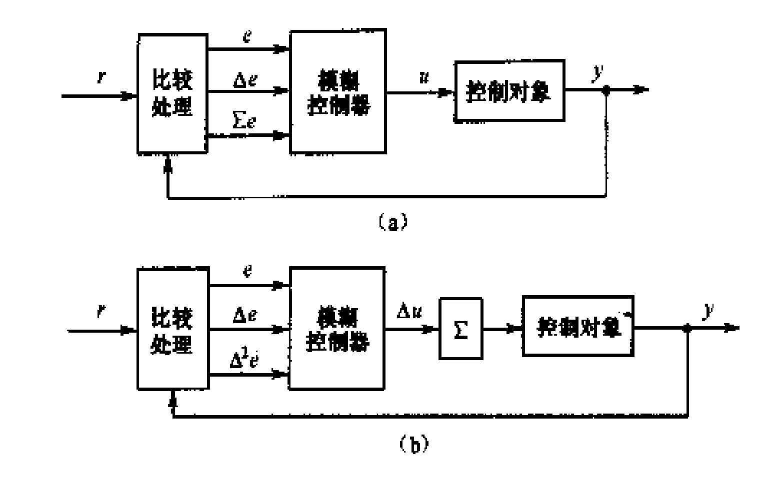


图 4-17 具有 PID 功能的模糊控制器的结构图

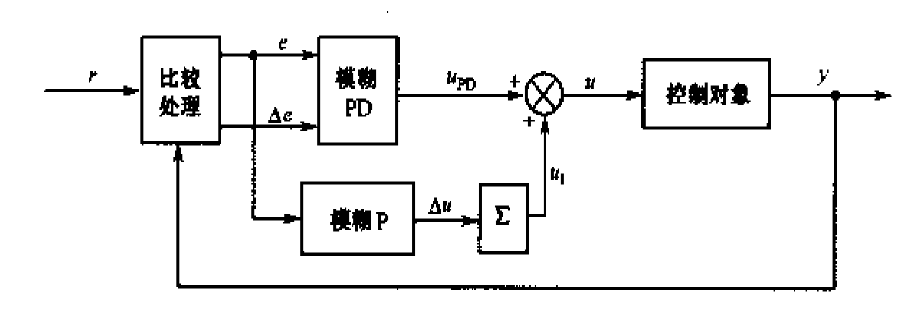


图 4-18 具有 PID 功能的模糊控制器的变形结构图

在如图 4-18 所示的变形结构中，采用两个模糊控制器。其中一个是最常见的具有 PD 功能的模糊控制器，简称为模糊 PD 控制器，它有两个输入，最多需 49 条规则（仍假设每个变量分 7 个等级）。另外一个是具有 P 功能的模糊控制器，简称为模糊 P 控制器，它只有一个输入，最多需 7 条规则。因此总共最多只需 49+7=56 条规则。可见，这种变形结构比通常的模糊 PD 控制器并未增加太大的复杂性，同时也实现了模糊 PID 控制器的功能。

最后指出，理论分析和实验都表明，只利用模糊控制器进行系统控制，往往不能满足控制对象的所有指标（尤其是在控制低层）。因此，一个完整的模糊控制系统还需要某种传统的控制器作为补充，一般采用的就是 PID 控制方法。通常，系统的控制器是由模糊控制器和常规 PID 控制器串联组成的。也就是说，PID 控制器的输入就是模糊控制器的输出，PID 控制器的输出就是整个控制器的输出。

第 5 章　MATLAB 模糊逻辑工具箱函数

针对模糊逻辑尤其是模糊控制的迅速推广应用，MathWorks 公司在其 MATLAB 版中添加了 Fuzzy Logic 工具箱。该工具箱由长期从事模糊逻辑和模糊控制研究与开发工作的有关专家和技术人员编制。MATLAB Fuzzy Logic 工具箱以其功能强大和方便易用的特点得到了用户的广泛欢迎。模糊逻辑的创始人 Zadeh 教授称赞该工具箱“在各方面都给人以深刻的印象，使模糊逻辑成为智能系统的概念与设计的有效工具”。

5.1　MATLAB 模糊逻辑工具箱简介

5.1.1　模糊逻辑工具箱的功能特点

1. 易于使用

模糊逻辑工具箱提供了建立和测试模糊逻辑系统的一整套功能函数，包括定义语言变量及其隶属度函数、输入模糊推理规则、整个模糊推理系统的管理以及交互式地观察模糊推理的过程和输出结果。

2. 提供图形化的系统设计界面

在模糊逻辑工具箱中包含五个图形化的系统设计工具。这五个设计工具为：

① 模糊推理系统编辑器。用于建立模糊逻辑系统的整体框架，包括输入与输出数目、去模糊化方法等；

② 隶属度函数编辑器。用于通过可视化手段建立语言变量的隶属度函数；

③ 模糊推理规则编辑器。用于建立模糊规则；

④ 系统输入输出特性曲面浏览器；

⑤ 模糊推理过程浏览器。

3. 支持模糊逻辑中的高级技术

① 自适应神经模糊推理系统（ANFIS，Adaptive Neural Fuzzy Inference System）；

② 用于模式识别的模糊聚类技术；

③ 模糊推理方法的选择，用户可在广泛采用的 Mamdani 型推理方法和 Takagi-Sugeno 型推理方法两者之间选择。

4. 集成的仿真和代码生成功能

模糊逻辑工具箱不但能够实现 Simulink 的无缝连接，而且通过 Real-Time Workshop 能

够生成 ANSI C 源代码，从而易于实现模糊系统的实时应用。

5．独立运行的模糊推理机

在用户完成模糊逻辑系统的设计后，可以将设计结果以 ASCII 码文件保存；利用模糊逻辑工具箱提供的模糊推理机，可以实现模糊逻辑系统的独立运行或者作为其他应用的一部分运行。

5.1.2　模糊推理系统的基本类型

在模糊推理系统中，模糊模型的表示主要有两类：一类是模糊规则的后件是输出量的某一模糊集合，如 NB、PB 等，这是最常用到的情况，因而称它为模糊系统的标准模型或 Mamdani 模型；另一类是模糊规则的后件是输入语言变量的函数，典型的情况是输入变量的线性组合。由于该方法是日本学者高木（Takagi）和关野（Sugeno）首先提出来的，因此通常称它为模糊系统的 Takagi-Sugeno（高木–关野）模型。

1．基于标准模型的模糊逻辑系统

在标准模型模糊逻辑系统中，模糊规则的前件和后件均为模糊语言值，即具有如下形式

$$\text{IF } x_1 \text{ is } \boldsymbol{A}_1 \text{ and } x_2 \text{ is } \boldsymbol{A}_2 \text{ and}\cdots\text{and } x_n \text{ is } \boldsymbol{A}_n \text{ THEN } y \text{ is } \boldsymbol{B}$$

式中，$\boldsymbol{A}_i(i=1,2,\cdots,n)$是输入模糊语言值；$\boldsymbol{B}$ 是输出模糊语言值。

基于标准模型的模糊逻辑系统的原理图如图 5-1 所示。图中的模糊规则库由若干“IF-THEN”规则构成。模糊推理在模糊推理系统中起着核心作用，它将输入模糊集合按照模糊规则映射成输出模糊集合。它提供了一种量化专家语言信息和在模糊逻辑原则下系统地利用这类语言信息的一般化模式。

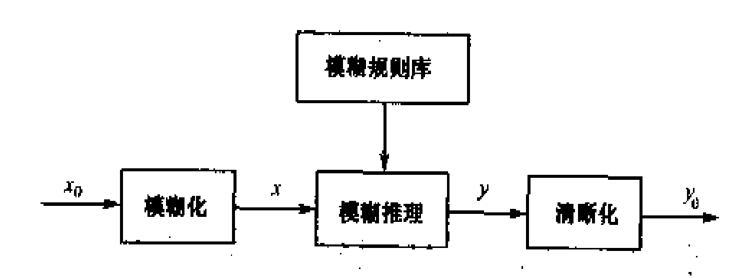


图 5-1　基于标准模型的模糊逻辑系统原理图

2．基于高木–关野（Takagi-Sugeno）模型的模糊逻辑系统

高木–关野模糊逻辑系统是一类较为特殊的模糊逻辑系统，其模糊规则不同于一般的模糊规则形式。在高木–关野模糊逻辑系统中，采用如下形式的模糊规则，即

$$\text{IF } x_1 \text{ is } \boldsymbol{A}_1 \text{ and } x_2 \text{ is } \boldsymbol{A}_2 \text{ and}\cdots\text{and } x_n \text{ is } \boldsymbol{A}_n \text{ THEN} \quad y=\sum_{i=1}^{n} c_i x_i$$

式中，$\boldsymbol{A}_i(i=1,2,\cdots,n)$是输入模糊语言值；$c_i(i=1,2,\cdots,n)$是真值参数。可以看出，高木–关野模糊逻辑系统的输出量 y 是精确值。这类模糊逻辑系统的优点是输出量可用输入值的线性组合

来表示，因而能够利用参数估计方法来确定系统的参数 $c_i(i=1,2,\cdots,n)$；同时，可以应用线性控制系统的分析方法来近似分析和设计模糊逻辑系统。其缺点是规则的输出部分不具有模糊语言值的形式，因此不能充分利用专家的控制知识，模糊逻辑的各种不同原则在这种模糊逻辑系统中应用的自由度也受到限制。

5.1.3 模糊逻辑系统的构成

前面讨论了模糊逻辑系统的基本类型，其中的标准型模糊逻辑系统应用最为广泛。在 MATLAB 模糊逻辑工具箱中，主要针对这一类型的模糊逻辑系统提供了分析和设计手段，但同时对高木–关野模糊逻辑系统也提供了一些相关函数。下面将以标准型模糊逻辑系统作为主要讨论对象。

构造一个模糊逻辑系统，首先必须明确其主要组成部分。一个典型的模糊逻辑系统主要由如下几个部分组成：

① 输入与输出语言变量，包括语言值及其隶属度函数；

② 模糊规则；

③ 输入量的模糊化方法和输出变量的去模糊化方法；

④ 模糊推理算法。

针对模糊逻辑系统的以上主要构成，在 MATLAB 模糊逻辑工具箱中构造一个模糊推理系统有如下步骤：

① 模糊推理系统对应的数据文件，其后缀为.fis，用于对该模糊系统进行存储、修改和管理；

② 确定输入、输出语言变量及其语言值；

③ 确定各语言值的隶属度函数，包括隶属度函数的类型与参数；

④ 确定模糊规则；

⑤ 确定各种模糊运算方法，包括模糊推理方法、模糊化方法、去模糊化方法等。

5.2 利用模糊逻辑工具箱建立模糊推理系统

5.2.1 模糊推理系统的建立、修改与存储管理

前面讨论了模糊推理系统的主要组成部分，即一个模糊推理系统由输入、输出语言变量及其隶属度函数、模糊规则、模糊推理机和去模糊化方法等各部分组成。在 MATLAB 模糊逻辑工具箱中，把模糊推理系统的各部分作为一个整体，并以文件形式对模糊推理系统进行建立、修改和存储等管理功能。表 5-1 给出了该工具箱提供的有关模糊推理系统管理的函数及其功能。

表5-1 模糊推理系统的管理函数及其功能

函数名	功能
newfis( )	创建新的模糊推理系统
readfis( )	从磁盘读出存储的模糊推理系统
getfis( )	获得模糊推理系统的特性数据
writefis( )	保存模糊推理系统
showfis( )	显示添加注释了的模糊推理系统
setfis( )	设置模糊推理系统的特性
plotfis( )	图形显示模糊推理系统的输入-输出特性

1. 创建新的模糊推理系统函数 newfis( )

该函数用于创建一个新的模糊推理系统，模糊推理系统的特性可由函数的参数指定，其参数个数可达7个。该函数的调用格式为

fisMat=newfis('fisName',fisType,andMethod,orMethod,impMethod,aggMethod,defuzzMethod)

式中，fisName 为模糊推理系统名称；fisType 为模糊推理类型；andMethod 为与运算操作符；orMethod 为或运算操作符；impMethod 为模糊蕴含方法；aggMethod 为各条规则推理结果的综合方法；defuzzMethod 为去模糊化方法；返回值 fisMat 为模糊推理系统对应的矩阵名称，模糊推理系统在 MATLAB 内存中数据是以矩阵形式存储的。

例 >>fisMat=newfis('newsys');getfis(fisMat)

显示：Name = newsys

Type = mamdani

NumInputs = 0

InLabels =

NumOutputs = 0

OutLabels =

NumRules = 0

AndMethod = min

OrMethod = max

ImpMethod = min

AggMethod = max

DefuzzMethod = centroid

2. 从磁盘中加载模糊推理系统函数 readfis( )

该函数的调用格式为

fisMat=readfis('filename')

式中，filename 为指定打开的模糊推理系统的数据文件名(.fis)，将其加载到当前的工作空间

（Workspace）中，当未指定文件名时，MATLAB 将会打开一个文件对话窗口，提示用户指定某一.fis 文件；返回值 fisMat 为模糊推理系统对应的矩阵名称。

例　>>fisMat=readfis('tipper'); getfis(fisMat)

显示：

```
Name = tipper
Type = mamdani
NumInputs = 2
InLabels = service
           food
NumOutputs = 1
OutLabels = tip
NumRules = 3
AndMethod = min
OrMethod = max
ImpMethod = min
AggMethod = max
DefuzzMethod = centroid
```

3. 获得模糊推理系统的属性函数 getfis( )

该函数的调用格式为

getfis(fisMat)

式中，fisMat 为模糊推理系统对应的矩阵名称。

4. 将模糊推理系统以矩阵形式保存在内存中的数据写入磁盘文件函数 writefis( )

模糊推理系统在内存中的数据是以矩阵形式存储的，其对应的矩阵名为 fisMat。当需要将模糊推理系统的数据写入磁盘文件时，就可利用 writefis( )函数，其调用格式为

writefis(fisMat)
writefis(fisMat,'filename')
writefis(fisMat,'filename','dialog')

式中，fisMat 为模糊推理系统对应的矩阵名称。在只有一个参数，即 writefis(fisMat)的情况下，MATLAB 将打开一个文件对话窗口，提示用户选择某一磁盘文件或输入一个新的文件名；用户也可直接在函数的第 2 个参数 filename 中指定某一磁盘文件名；writefis(fisMat,'filename','dialog')则打开一个以 filename 为默认文件名的对话窗口，用户仍可重新输入文件名。文件名的后缀默认为.fis。

例　>>fisMat=newfis('tipper');

>>fisMat=addvar(fisMat,'input', 'service',[0　10]);

>>fisMat=addmf(fisMat, 'input',1, 'poor', 'gaussmf',[1.5　0]);

```
>>fisMat=addmf(fisMat, 'Input',1, 'good', 'gaussmf',[1.5  5]);
>>fisMat=addmf(fisMat, 'input',1, 'excellent', 'gaussmf',[1.5  10]);
>>writefis(fisMat, 'my_ file')。
```

5. 以分行的形式显示模糊推理系统矩阵的所有属性函数 showfis( )

该函数的调用格式为

showfis(fisMat)

式中，fisMat 为模糊推理系统对应的矩阵名称。

例 >>fisMat=readfis('tipper');showfis(fisMat)

6. 设置模糊推理系统的属性函数 setfis( )

该函数的调用格式为

```
fisMat=setfis(fisMat, 'propname',newprop)
fisMat=setfis(fisMat,vartype,varindex, 'propname',newprop)
fisMat=setfis(fisMat,vartype,varindex, 'mf',mfindex, 'propname',nemeprop);
```

该函数的参数个数可以有3、5、7三种情况。当参数个数为3时，用于设定模糊推理系统的全局属性，包括：name 为模糊推理系统的名称；type 为模糊推理系统的类型；numinputs 为模糊推理系统的输入变量个数；numoutputs 为模糊推理系统的输出变量个数；numrules 为规则个数；andmethod 为与运算方法；ormethod 为或运算方法；impmethod 为模糊蕴含方法；aggmethod 为各个规则推理结果的综合方法； defuzzmethod 为输出去模糊化方法。

当参数个数为 5 个时，用于设定模糊推理系统矩阵某一个语言变量的属性，这些属性包括：name（变量名称），bounds（论域范围）。

当参数变量为 7 时，用于设定一个语言变量的某一隶属度函数的属性，包括：name（隶属度函数名称），type（类型），params（参数）。

例如，七个参数的情况

```
>>fisMat1=setfis(fisMat, 'input',1, 'mf',2, 'name', 'wretched');
>>getfis(fisMat1, 'input',1, 'mf',2, 'name')
```

5.2.2 模糊语言变量及其语言值

在模糊推理系统中，专家的控制知识以模糊规则形式表示。为直接反映人类自然语言的模糊性特点，模糊规则的前件和后件中引入语言变量和语言值的概念。语言变量分为输入语言变量和输出语言变量：输入语言变量是对模糊推理系统输入变量的模糊化描述，通常位于模糊规则的前件中；输出语言变量是对模糊推理系统输出变量的模糊化描述，通常位于模糊规则的后件中。语言变量具有多个语言值，每个语言值对应一个隶属度函数。语言变量的语言值构成了对输入和输出空间的模糊分割，模糊分割的个数，即语言值的个数以及语言值对应的隶属度函数决定了模糊分割的精细化程度。模糊分割的个数也决定了模糊规则的个

数，模糊分割数越多，控制规则数也越多。因此，在设计模糊推理系统时，应在模糊分割的精细程度与控制规则的复杂性之间取得折中。

在MATLAB模糊逻辑工具箱中，提供了向模糊推理系统添加或删除模糊语言变量及其语言值的函数，见表5-2。

表5-2 添加或删除模糊语言变量函数

函数名	功能
addvar( )	添加模糊语言变量
rmvar( )	删除模糊语言变量

1. 向模糊推理系统添加语言变量函数 addvar( )

该函数的调用格式为

fisMat2=addvar(fisMat1,varType,varName,varBounds)

式中，fisMat1/2 为模糊推理系统的对应矩阵名称；varType 用于指定语言变量的类型；varName 用于指定语言变量的名称；varBounds 用于指定变量的论域范围。对于添加到同一个模糊推理系统的语言变量，将按照添加的先后顺序分别赋予一个编号，编号从1开始，逐渐递增。对输入与输出语言变量，则独立地分开编号。

例

```
>>fisMat=newfis('tipper');
>>fisMat=addvar(fisMat, 'input', 'service',[0 10]);getfis(fisMat, 'input',1)
```

显示：

```
Name = service
NumMFs = 3
MFLabels =
Range = [0 10]
```

2. 从模糊推理系统中删除语言变量 rmvar( )

该函数的调用格式为

fisMat2=rmvar(fisMat1, 'varType',varIndex)

式中，fisMat1/2 为模糊推理系统的对应矩阵名称；varType 用于指定语言变量的类型。

当一个模糊语言变量正在被当前的模糊规则集使用时，则不能删除该变量。在一个模糊语言变量被删除后，MATLAB 模糊逻辑工具箱将会自动地对模糊规则集进行修改，以保证一致性。

例

```
>>fisMat=newfis('mysys');fisMat=addvar(fisMat, 'input', 'temperature',[0 100]);
>>getfis (fisMat)
```

显示：

```
Name      = mysys
Type      = mamdani
NumInputs = 1
```

```
        InLabels   =   temperature
        NumOutputs = 0
        OutLabels =
        NumRules = 0
        AndMethod = min
        OrMethod = max
        ImpMethod = min
        AggMethod = max
        DefuzzMethod = centroid
```

例 >>B=rmvar(fisMat, 'input',1);getfis(B)

显示：

```
        Name = mysys
        Type = mamdani
        NumInputs = 0
        InLabels   =
        NumOutputs = 0
        OutLabels =
        NumRules = 0
        AndMethod = min
        OrMethod = max
        ImpMethod = min
        AggMethod = max
        DefuzzMethod = centroid
```

5.2.3 模糊语言变量的隶属度

每个模糊语言变量具有多个模糊语言值。模糊语言值的名称通常具有一定的含义，如 NB（负大）、NM（负中）、NS（负小）、ZE（零）、PS（正小）、PM（正中）、PB（正大）等。每个语言值都对应一个隶属度函数。隶属度函数可有两种描述方式，即数值描述方式和函数描述方式。数值描述方式适用于语言变量的论域为离散的情形，此时隶属度函数可用向量或表格的形式来表示；对于论域为连续的情况，隶属度函数则采用函数描述方式。

在 MATLAB 模糊逻辑工具箱中支持的隶属度函数类型有如下几种，即高斯型、三角型、梯型、钟型、Sigmoid 型、π型和 Z 型。利用工具箱中提供的函数可以建立和计算上述各种类型隶属度函数。

隶属度函数曲线的形状决定了对输入、输出空间的模糊分割，对模糊推理系统的性能有重要的影响。在 MATLAB 模糊逻辑工具箱中提供了丰富的隶属度函数类型的支持，利用

工具箱的有关函数可以方便地对各类隶属度函数进行建立、修改和删除等操作。语言变量的隶属度函数见表5-3。

表5-3 语言变量的隶属度函数

函数名	功能
plotmf( )	绘制隶属度函数曲线
addmf( )	添加模糊语言变量的隶属度函数
rmmf( )	删除隶属度函数
gaussmf( )	建立高斯型隶属度函数
gauss2mf( )	建立双边高斯型隶属度函数
gbellmf( )	建立一般的钟型隶属度函数
pimf( )	建立π型隶属度函数
sigmf( )	建立Sigmiod型的隶属度函数
trapmf( )	建立梯型隶属度函数
trimf( )	建立三角型隶属度函数
zmf( )	建立Z型隶属度函数
mf2mf( )	隶属度函数间的参数转换
psigmf( )	计算两个Sigmiod型隶属度函数之积
dsigmf( )	计算两个Sigmiod型隶属度函数之和

1. 绘制语言变量的隶属度曲线函数plotmf( )

该函数的调用格式为

plotmf(fisMat,varType,varIndex)

式中，fisMat 为模糊推理系统的对应矩阵名称；varType 为语言变量的类型；varIndex 为语言变量的编号。

2. 向模糊推理系统的语言变量添加隶属度函数addmf( )

隶属度函数只能给作为模糊推理系统中存在的某一语言变量的语言值添加，而不能添加到一个不存在的语言变量中。某个语言变量的隶属度函数（即语言值）按照添加的顺序加以编号，第一个添加的隶属度函数被编为1号，此后依次递增。该函数调用格式为

fisMat2=addmf(fisMat1,varType,varIndex,mfName,mfType,mfParams)

式中，fisMat1/2 为模糊推理系统的对应矩阵；varType 指定语言变量的类型（输入或输出）；varIndex 指定语言变量的编号；mfName 指定隶属度函数的名称；mfType 和 mfParams 分别指定隶属度函数的类型和参数。

例如，利用以下命令可得如图5-2所示的隶属度函数曲线。

```
>>fisMat=newfis('tipper ');
>>fisMat=addvar(fisMat,'input','service',[0  10]);
```

```
>>fisMat=addmf(fisMat,'input',1,'poor','gaussmf',[1.5  0 ]);
>>fisMat=addmf(fisMat,'input',1,'good','gaussmf',[1.5  5]);
>>fisMat=addmf(fisMat,'input',1,'excellent','gaussmf',[1.5 10]);
>>plotmf(fisMat,'input',1)。
```

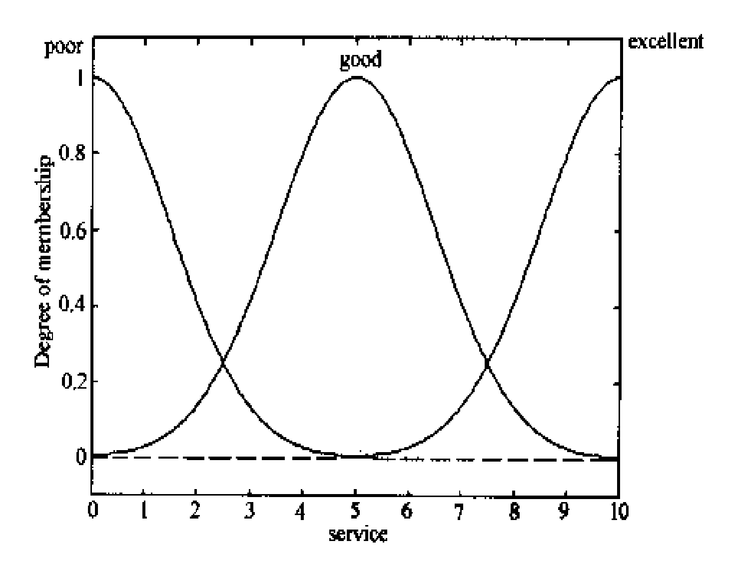


图 5-2　隶属度函数曲线

3. 从模糊推理系统中删除一个语言变量的某一隶属度函数 rmmf( )

当一个隶属度函数正在被当前模糊推理规则使用时，则不能删除。该函数的调用格式为

fisMat2=rmmf(fisMat1,varType,varIndex,' mf ',mfIndex)

式中，fisMat1/2 为模糊推理系统的对应矩阵；varType 为语言变量的类型；varIndex 为语言变量的编号；mf 为隶属度函数的名称；mfIndex 为隶属度函数的编号。

4. 建立高斯型隶属度函数 gaussmf( )

该函数的调用格式为

y=gaussmf(x,params)

y=gaussmf(x,[sig c])

式中，c 决定了函数的中心点；sig 决定了函数曲线的宽度σ。高斯型函数的形状由σ 和 c 两个参数决定，高斯函数的表达式如下

$$y=e^{-\frac{(x-c)^2}{\sigma^2}}$$

参数 x 用于指定变量的论域。

例　利用以下命令，可建立如图 5-3 所示的高斯型隶属度函数曲线。

```
>>x=0:0.1:10;y=gaussmf(x,[2  5]);
>>plot(x,y);
```

```
>>xlabel('gaussmf, p=[2   5]')。
```

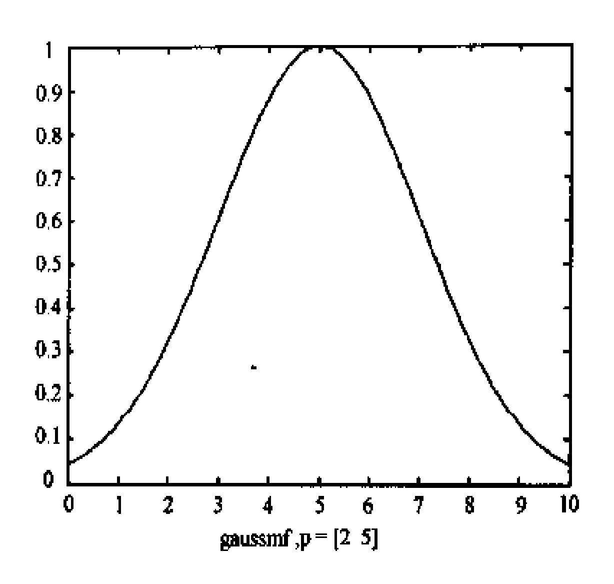


图5-3 高斯型隶属度函数曲线

5．建立双边高斯型隶属度函数 gauss2mf( )

该函数的调用格式为

y=gauss2mf(x,params)

y=gauss2mf(x,[sig1 c1 sig2 c2])

双边高斯型函数的曲线由两个中心点不相同的高斯型函数的左、右半边曲线组合而成，其表达式如下。参数 sigl,c_1,sig2,c_2 分别对应左、右半边高斯函数的宽度与中心点，$c_2>c_1$。

$$y=\begin{cases} e^{-\frac{(x-c_1)^2}{\sigma_1^2}}, & x<c_1 \\ e^{-\frac{(x-c_2)^2}{\sigma_2^2}}, & x\geqslant c_2 \end{cases}$$

例 利用以下命令，可建立如图5-4所示的双边高斯型隶属度函数。

```
>>x=0:0.1:10;
>>y=gauss2mf(x, [1   3    3   4]);
>>plot(x,y);
>>xlabel('gauss2mf, p=[1   3   3   4]')。
```

6．建立一般的钟型隶属度函数 gbellmf( )

该函数的调用格式为

y=gbellmf(x,params)

y=gbellmf(x,[a b c])

式中，参数 x 指定变量的论域范围；[a b c]指定钟型函数的形状。钟型函数的表达式为

$$y=\frac{1}{1+\left|\dfrac{x-c}{a}\right|^{2b}}$$

例如，利用以下命令可建立如图 5-5 所示的钟型隶属度函数曲线。

```
>>x=0:0.1:10;
>>y=gbellmf(x, [2  4  6]);
>>plot(x,y);
>>xlabel('gbellmf, p=[2  4  6]')。
```

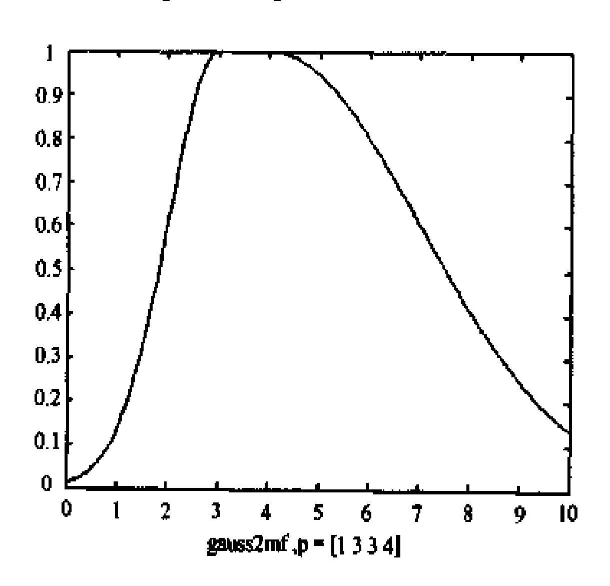


图 5-4　双边高斯型隶属度函数曲线

gbellmf,P = [2 4 6]

图 5-5　钟型隶属度函数曲线

7．建立π型隶属度函数 pimf( )

π型函数是一种基于样条的函数，因其形状类似字母π而得名。该函数的调用格式为

y=pimf(x,params)

y=pimf(x,[a b c d])

式中，参数 x 指定函数的自变量范围；[a b c d] 决定函数的形状，a 和 d 分别对应曲线下部的左右两个拐点，b 和 c 分别对应曲线上部的左右两个拐点。

例如，利用以下命令可建立如图 5-6 所示的π型隶属度函数曲线。

```
>>x=0:0.1:10;y=pimf(x, [1 4 5 10 ]);
>>plot(x,y),xlabel('pimf,p=[1 4 5 10]')
```

8．建立 Sigmoid 型隶属度函数 sigmf( )

该函数的调用格式为

y=sigmf(x,params)

y=sigmf(x,[a c])

式中，参数 x 用于指定变量的论域范围；[a　c]决定了 Sigmoid 型函数的形状。其表达式为

$$y=\frac{1}{1+\mathrm{e}^{-a(x-c)}}$$

Sigmoid 型函数曲线具有半开的形状，因而适于作为“极大”、“极小”等语言值的隶属度函数。

例如，利用以下命令可建立如图 5-7 所示的 Sigmoid 型隶属度函数曲线。

```
>>x=0:0.1:10;y=sigmf(x, [2   4 ]);
>>plot(x,y),xlabel('sigmf, p=[2   4 ]')。
```

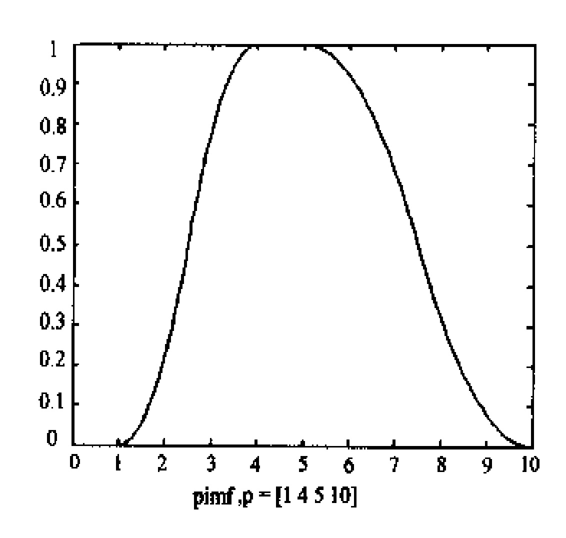


图 5-6　π型隶属度函数曲线

图 5-7　Sigmoid 型隶属度函数曲线

9. 建立梯型隶属度函数 trapmf( )

该函数的调用格式为

```
y=trapmf(x,params)
y=trapmf(x,[a,b,c,d])
```

式中，参数 x 指定变量的论域范围；参数 a、b、c 和 d 指定梯型隶属度函数的形状。其对应的表达式为

$$f(x,a,b,c,d)=\begin{cases}0 & x<a\\ \dfrac{x-a}{b-a} & a\leqslant x\leqslant b\\ 1 & b<x<c\\ \dfrac{d-x}{d-c} & c\leqslant x\leqslant d\\ 0 & d<x\end{cases}$$

例如，利用以下命令可建立如图 5-8 所示的梯型隶属度函数曲线。

```
>>x=0:0.1:10;y=trapmf(x,[1   5   7   8]);
>>plot(x,y),xlabel('trapmf,p=[1   5   7   8]')
```

10. 建立三角型隶属度函数 trimf( )

该函数的调用格式为

y=*trimf*(x,*params*)

y=trimf(x,[a,b,c])

式中，参数 x 指定变量的论域范围；参数 a、b 和 c 指定三角型函数的形状，其表达式为

$$f(x,a,b,c,d)=\begin{cases}0 & x<a \\ \dfrac{x-a}{b-a} & a\leqslant x\leqslant b \\ \dfrac{c-x}{c-b} & b<x\leqslant c \\ 0 & c<x\end{cases}$$

例如，利用以下命令可建立如图 5-9 所示的三角型隶属度函数并绘制曲线。

```
>>x=0:0.1:10;y=trimf(x,[3  6  8]);
>>plot(x,y),xlabel('trimf,p=[3  6  8]')。
```

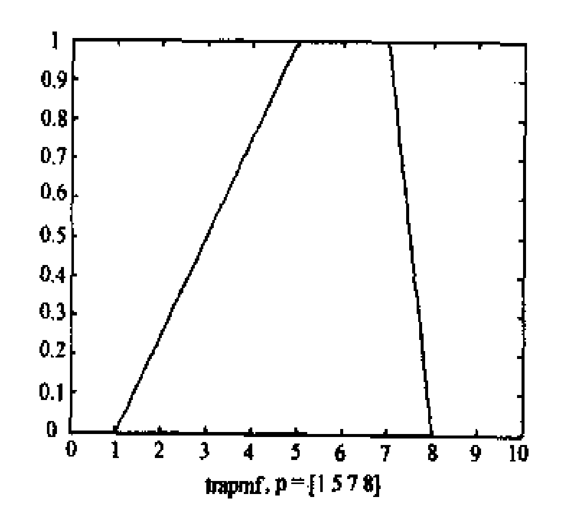


图 5-8 梯型隶属度函数曲线

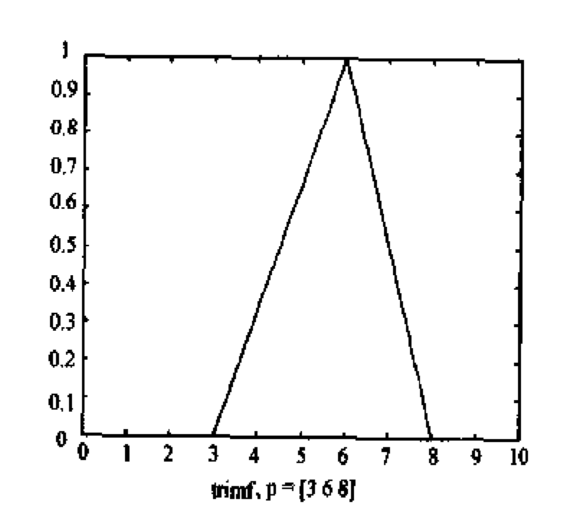


图 5-9 三角型隶属度函数

11. 建立 Z 型隶属度函数 zmf( )

该函数的调用格式为

y=zmf(x,params)

y=zmf(x,[a,b])

Z 型函数是一种基于样条插值的函数，两个参数 a 和 b 分别定义样条插值的起点和终点；参数 x 指定变量的论域范围。

例如，利用以下命令可建立如图 5-10 所示的 Z 型隶属度函数曲线。

```
>>x=0:0.1:10;y= zmf(x,[3 6]);
>>plot(x,y),xlabel('zmf,p=[3 6]')。
```

12. 通过两个 Sigmoid 型函数的乘积来构造新的隶属度函数 psigmf( )

为了得到更符合人们习惯的隶属度函数形状，可以利用两个 Sigmoid 型函数之和或乘积来构造新的隶属度函数类型，模糊逻辑工具箱中提供了相应的函数。该函数的调用格式为

y=psigmf(x,params)

y=psigmf(x,[a1 c1 a2 c2])

式中，参数 a1、c1 和 a2、c2 分别用于指定两个 Sigmoid 型函数的形状；参数 x 指定变量的使用范围。新的函数表达式为

$$y=\frac{1}{(1+e^{-a_1(x-c_1)})(1+e^{-a_2(x-c_2)})}$$

例如，利用以下命令，由两个 Sigmoid 型函数的乘积来构造新的隶属度函数，如图 5-11 所示。

```
>>x=0:0.1:10;y=psigmf(x,[2  3  -5  8]);
>>plot(x,y);
>>xlabel('psigmf,p=[2  3  -5  8]')。
```

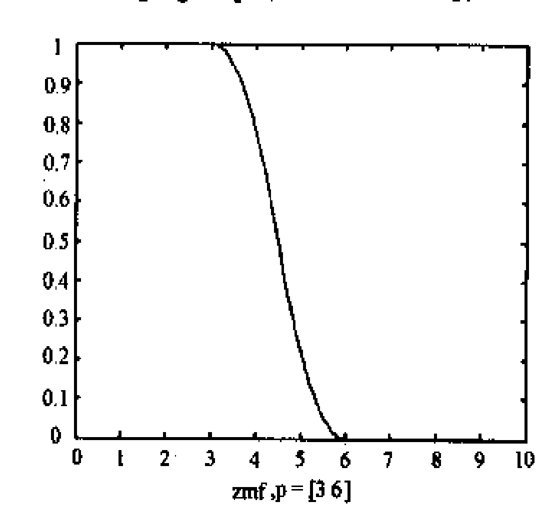


图 5-10 Z 型隶属度函数曲线

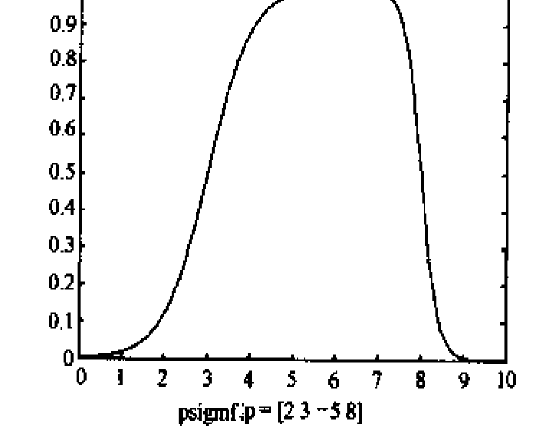


图 5-11 两个 Sigmoid 型函数的乘积

13. 通过计算两个 Sigmoid 型函数之和来构造新的隶属度函数 dsigmf( )

该函数的调用格式为

y=dsigmf(x,params)

y=dsigmf(x,[a1,c1,a2,c2])

该函数的用法与函数 psigmf( )类似，参数 a1、c1 和 a2、c2 分别用于指定两个 Sigmoid 型函数的形状，构造得到的新的隶属度函数表达式为

$$y=\frac{1}{1+e^{-a_1(x-c_1)}}+\frac{1}{1+e^{-a_2(x-c_2)}}$$

例如，利用以下命令可绘制两个 Sigmoid 型函数之和的隶属度函数曲线，如图 5-12 所示。

```
>>x=0:0.1:10;
>>y=dsigmf(x,[5   2   5   7]);
>>plot(x,y);
>>xlabel('dsigmf,p=[5   2   5   7]')。
```

14．进行不同类型隶属度函数之间的参数转换函数 mf2mf( )

该函数的调用格式为

outParams=mf2mf(inParams,inType,outType)

式中，inParams 为转换前的隶属度函数的参数；outParams 为转换后的隶属度函数的参数；inType 为转换前的隶属度函数的类型；outType 为转换后的隶属度函数的类型。

该函数将尽量保持两种类型的隶属度函数曲线在形状上的近似，特别是保持隶属度等于 0.5 处的点的重合。但不可避免地会丢失一些信息。因此，当再次使用该函数进行反向转换时，将无法得到与原来函数相同的参数。

例如，利用以下命令可实现钟型隶属度函数向三角型隶属度函数的转换，如图 5-13 所示。

```
>>x=0:0.1:5;mfp1=[1 2 3];
>>mfp2=mf2mf(mfp1,'gbellmf','trimf');
>>plot(x,gbellmf(x,mfp1),x,trimf(x,mfp2))
```

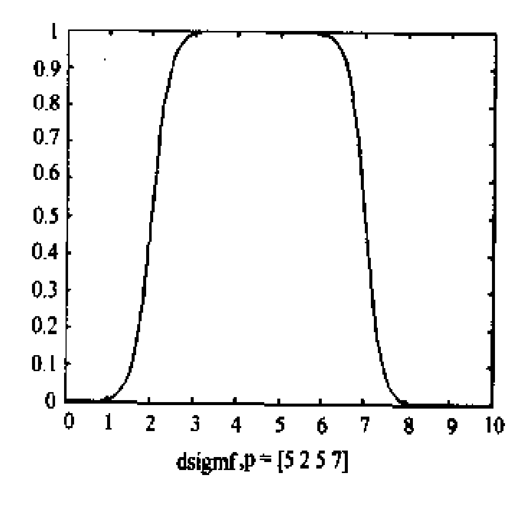
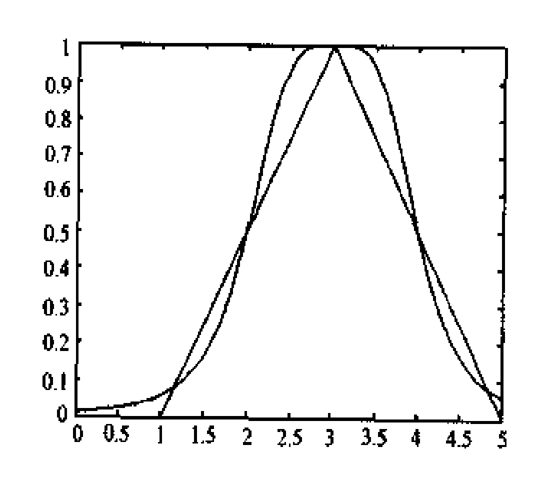



图 5-12　两个 Sigmoid 型函数之和


图 5-13　隶属度函数的转换

5.2.4　模糊规则的建立与修改

在模糊推理系统中，模糊规则以模糊语言的形式描述人类的经验和知识，规则是否正确地反映人类专家的经验和知识，是否反映对象的特性，直接决定模糊推理系统的性能。通常，模糊规则的形式是"IF 前件 THEN 后件"。前件由对模糊语言变量的语言值描述构成，如"温度较高，压力较低"。在一般的模糊推理系统中，后件由对输出模糊语言变量的语言

值描述构成，但在高木–关野模糊推理系统中，后件将输出变量表示成输入量的精确值的组合。模糊规则的这种形式化表示是符合人们通过自然语言对许多知识的描述和记忆习惯的。

模糊规则的建立是构造模糊推理系统的关键。在实际应用中，初步建立的模糊规则往往难以达到良好的效果，必须不断加以修正和试凑。在模糊规则的建立修正和试凑过程中，应尽量保证模糊规则的完备性和相容性。在 MATLAB 模糊逻辑工具箱中，提供了有关对模糊规则建立和修改的函数，见表 5-4。

表 5-4 模糊规则建立和修改的函数

函 数 名	功 能
addrule( )	向模糊推理系统添加模糊规则函数
parsrule( )	解析模糊规则函数
showrule( )	显示模糊规则函数

1．向模糊推理系统添加模糊规则函数 addrule( )

该函数的调用格式为

fisMat2=addrule(fisMat1,rulelist)

式中，参数 fisMat1/2 为模糊推理系统对应的矩阵名称；rulelist 以向量的形式给出需要添加的模糊规则，该向量的格式有严格的要求，如果模糊推理系统有 m 个输入语言变量和 n 个输出语言变量，则向量 rulelist 的列数必须为 m+n+2，而行数任意。在 rulelist 的每一行中，前 m 个数字表示各输入语言变量的语言值，其后的 n 个数字表示输出语言变量的语言值，第 m+n+1 个数字是该规则适用的权重，权重的值在 0 到 1 之间，一般设定为 1；第 m+n+2 个数字为 0 或 1 两个值之一，若为 1，则表示模糊规则前件的各语言变量之间是“与”的关系，若是 0，则表示是“或”的关系。

例
```
>>rulelist=[1  1  1  1  1;1  2  2  1  1];
>>fisMat=addrule(fisMat,rulelist)
```
显示：
```
          name: 'tipper'
          type: 'mamdani'
     andMethod: 'min'
      orMethod: 'max'
  defuzzMethod: 'centroid'
     impMethod: 'min'
     aggMethod: 'max'
         input: [1x1 struct]
        output: []
          rule: [1x2 struct]
```

2. 解析模糊规则函数 parsrule( )

函数 parsrulc( )对给定的模糊语言规则进行解析并添加到模糊推理系统矩阵中，其调用格式为

fisMat2=parsrule(fisMat1,txtRuleList,ruleFormat)

式中，fisMat1/2 为模糊推理系统矩阵；txtRuleList 为模糊语言规则；ruleFormat 为规则的格式，包括语言型（'verbose'）、符号型（'symbolic'）和索引型（'indexed'）。

例　>>fisMat=readfis('tipper');ruleTxt='if service is poor then tip is generous';

>>fisMat2=parsrule(fisMat,ruleTxt,'verbose');showrule(fisMat2)

输出为

1. If (service is poor) then (tip is generous) (1)

3. 显示模糊规则函数 showrule( )

该函数的调用格式为

showrule(fisMat,indexList,format)

该函数用于显示指定的模糊推理系统的模糊规则，模糊规则可以按三种方式显示，即：语言方式（verbose）、符号方式（symbolic）和隶属度函数编号方式（membership function index referencing）。第一个参数是模糊推理系统矩阵的名称，第二个参数是规则编号，第三个参数是规则显示方式。规则编号可以以向量形式指定多个规则。

例　>>fisMat=readfis('tipper'); showrule(fisMat,1)

其输出结果为

1. If (service is poor) or (food is rancid) then (tip is cheap) (1)

例　>>showrule(fisMat,2)

其输出结果为

2. If (service is good) then (tip is average) (1)

例　>>showrule(fisMat,[3 1],'symbolic')

其输出结果为

3. (service==excellent) | (food==delicious) => (tip=generous) (1)

1. (service==poor) | (food==rancid) => (tip=cheap) (1)

例　>>showrule(fisMat,1:3,'indexed')

其输出结果为

```
ans =
     1 1, 1 (1) : 2
     2 0, 2 (1) : 1
     3 2, 3 (1) : 2
```

5.2.5 模糊推理计算与去模糊化

在建立好模糊语言变量及其隶属度的值并构造完成模糊规则之后，就可执行模糊推理计算了。模糊推理的执行结果与模糊蕴含操作的定义、推理合成规则、模糊规则前件部分的连接词“and”的操作定义等有关，因而有多种不同的算法。

目前常用的模糊推理合成规则是“极大–极小”合成规则，设 $\boldsymbol{R}$ 表示规则：“$\boldsymbol{X}$ 为 $\boldsymbol{A}\to\boldsymbol{Y}$ 为 $\boldsymbol{B}$”表达的模糊关系，则当 $\boldsymbol{X}$ 为 $\boldsymbol{A}'$时，按照“极大–极小”规则进行模糊推理的结论 $\boldsymbol{B}'$计算为

$$\boldsymbol{B}'=\boldsymbol{A}'\circ\boldsymbol{R}=\int_Y \bigvee_{x\in X}(\mu_{A'}(x)\wedge\mu_R(x,y))/y$$

基于模糊蕴含操作的不同定义，人们提出了多种模糊推理算法，其中较为常用的是 Mamdani 模糊推理算法和 Larsen 模糊推理算法。另外，对于输出为精确量的一类特殊模糊逻辑系统——Takagi-Sugeno 型模糊推理系统，采用了将模糊推理与去模糊化结合的运算操作。与其他类型的模糊推理方法不同，Takagi-Sugeno 型模糊推理将去模糊化也结合到模糊推理中，其输出为精确量。这是由 Takagi-Sugeno 型模糊规则的形式所决定的，在 Takagi-Sugeno 型模糊规则的后件部分将输出量表示为输入量的线性组合，零阶 Takagi-Sugeno 型模糊规则具有如下形式，即

$$\text{IF}\quad x\text{为}\boldsymbol{A}\text{ 且 }y\text{为}\boldsymbol{B}\quad\text{THEN}\quad z=k$$

式中，k 为常数。而一阶 Takagi-Sugeno 型模糊规则的形式为

$$\text{IF}\quad x\text{为}\boldsymbol{A}\text{ 且 }y\text{为}\boldsymbol{B}\quad\text{THEN}\quad z=p*x+q*y+r$$

式中，p、q、r 均为常数。

对于一个由 n 条规则组成的 Takagi-Sugeno 型模糊推理系统，设每条规则具有下面的形式：

$$\boldsymbol{R}_i\text{: IF }x\text{为}\boldsymbol{A}_i\text{且 }y\text{为}\boldsymbol{B}_i\quad\text{THEN}\quad z=z_i\qquad(i=1,2,\cdots,n)$$

则系统总的输出用下式计算

$$y=\frac{\sum_{i=1}^{n}\mu_{A_i}(x)\mu_{B_i}(y)z_i}{\sum_{i=1}^{n}\mu_{A_i}(x)\mu_{B_i}(y)}$$

在 MATLAB 模糊逻辑工具箱中提供了有关对模糊推理计算与去模糊化的函数，见表 5-5。

表 5-5 模糊推理计算与去模糊化的函数

函 数 名	功 能
evalfis( )	执行模糊推理计算函数
defuzz( )	执行输出去模糊化函数
gensurf( )	生成模糊推理系统的输出曲面并显示函数

1．执行模糊推理计算函数 evalfis( )

该函数的调用格式为

```
output=evalfis(input,fisMat)
```

该函数计算以 input 为输入模糊向量的模糊推理系统的输出模糊向量 output。fisMat 为模糊推理系统的矩阵名称。evalfis( )有两种文件格式，即 m-文件和 MEX-文件。考虑到运算的速度，通常调用 MEX-文件执行模糊推理计算。输入向量是 M×N 矩阵，其中的 N 是输入变量个数；输出向量是 M×L 矩阵，其中的 L 是输出变量个数。

例

```
>>fisMat=readfis('tipper');
>>out=evalfis([2 1;4 9],fisMat)
```

其输出结果为

```
out =
   7.0169
  19.6810
```

2．执行输出去模糊化函数 defuzz( )

该函数的调用格式为

```
out=defuzz(x,mf,'type')
```

式中，参数 x 是变量的论域范围；mf 为待去模糊化的模糊集合；type 是去模糊化的方法，去模糊化的方法包括 5 种，即 centroid（面积中心法）、bisector（面积平分法）、mom（平均最大隶属度方法）、som（最大隶属度中的取最小值方法）和 lom（最大隶属度中的取最大值方法）。

例

```
>>x=-10:0.1:10;
>>mf=trapmf(x,[-10 -8 -4 7]);
>>xx=defuzz(x,mf,'centroid')
```

其输出结果为

```
xx =
   -3.2857
```

3．生成模糊推理系统的输出曲面并显示函数 gensurf( )

该函数的调用格式为

```
gensurf(fisMat)
gensurf(fisMat,inputs,outputs)
gensurf(fisMat,inputs,outputs,grids,refinput)
```

式中，参数 fisMat 为模糊推理系统对应的矩阵；inputs 为模糊推理系统的一个或两个输入语言变量；output 为模糊系统的输出语言变量；参数 grids 用于指定 x 和 y 坐标方向的网络数

目：当系统输入变量多于两个时，参数 refinput 用于指定保持不变的输入变量；仅有一个参数 fisMat 时，该函数生成由模糊推理系统的前两个输入和第一个输出构成的三维曲面。

例如，利用以下命令可得如图 5-14 所示的输出曲面。

```
>> fisMat=readfis('tipper') ;gensurf(fisMat)
```

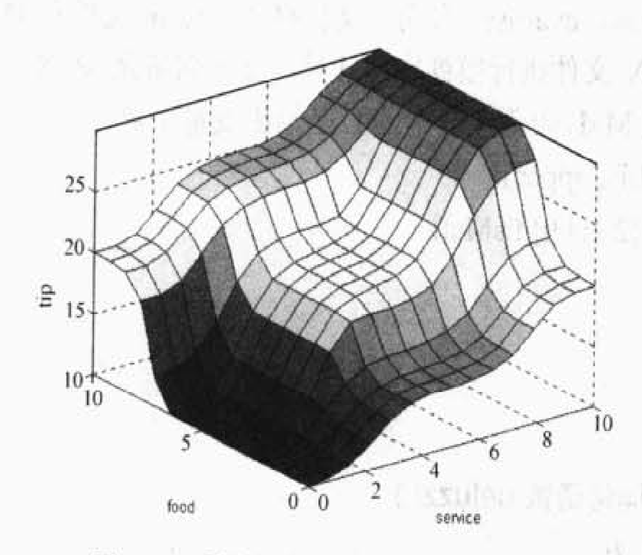


图 5-14 模糊推理系统输出特性曲面图

5.3 MATLAB 模糊逻辑工具箱的图形用户界面

前面介绍了模糊逻辑工具箱中有关构造模糊推理系统的函数，这些函数都是在 MATLAB 命令行窗口执行并显示结果的。为了进一步方便用户，模糊逻辑工具箱提供了一套用于构造模糊推理系统的图形用户界面，它具有以下五大功能。

5.3.1 模糊推理系统编辑器(Fuzzy)

模糊推理系统编辑器提供了利用图形界面（GUI）对模糊系统的高层属性的编辑和修改功能，这些属性包括输入、输出语言变量的个数和去模糊化方法等。用户在基本模糊编辑器中，可以通过菜单选择激活其他几个图形界面编辑器，如隶属度函数编辑器（mfedit）、模糊规则编辑器（ruleedit）等。

在 MATLAB 命令窗口中，可以用以下两种方法启动模糊推理系统编辑器 FIS Editer：

① 在 MATLAB 的命令窗口中直接键入 fuzzy 命令；

② 在 MATLAB 的 Launch Pad 窗口中，用鼠标双击模糊逻辑系统工具箱(Fuzzy Logic Toolbox)中的 FIS Editor Viewer 项。

在以上两种启动方式下，模糊推理系统编辑器的图形界面如图 5-15 所示。

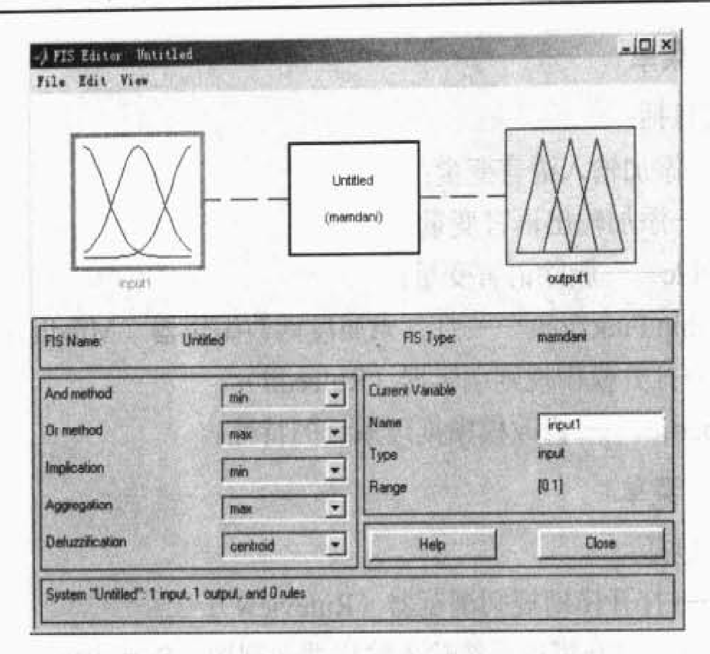


图 5-15 模糊推理系统编辑器图形界面

从图 5-15 中可以看到，在窗口上半部，以图形框的形式列出了模糊推理系统的基本组成部分，即输入模糊变量（input1）、模糊规则（mamdani）和输出模糊变量（output1）。通过鼠标双击上述图形框，能够激活隶属度函数编辑器和模糊规则编辑器等相应的编辑窗口。在窗口的下半部分的左侧，列出了模糊推理系统的名称、类型和一些基本属性，包括“与”运算（And method）、“或”运算（Or method）、蕴含运算（Implication）、模糊规则的综合运算（Aggregation）以及去模糊化（Defuzzification）等。用户只需用鼠标即可设定相应的属性。在图 5-15 中，模糊推理系统的基本属性默认设定为：“与”运算采用极小运算（min）；“或”运算采用极大运算（max）；模糊蕴含采用极小运算（min）；模糊规则综合采用极大运算（max）；去模糊化采用重心法（centroid）。在窗口下半部分的右侧，列出了当前选定的模糊语言变量（Current Variable）的名称、类型及其论域范围。

模糊推理系统编辑器的菜单部分主要提供了如下功能。

1. 文件（File）菜单

文件菜单的主要功能包括：

① New Mamdani FIS——新建 Mamdani 型模糊推理系统；

② New Sugeno FIS——新建 Sugeno 型模糊推理系统；

③ Load FIS from Workspace——从 MATLAB 工作空间加载一个模糊推理系统；

④ Load FIS From Disk——从磁盘打开一个模糊推理系统文件；

⑤ Save to Workspace——将当前的模糊推理系统保存到 MATLAB 工作空间；

⑥ Save to disk——将当前的模糊推理系统保存到磁盘文件；

⑦ Print——打印模糊推理系统的信息；

⑧ Close window——关闭窗口。

2. 编辑（Edit）菜单

编辑菜单的功能包括：

① Add input——添加输入语言变量；

② Add output——添加输出语言变量；

③ Remove variable——删除语言变量；

④ Edit Membership Functions——打开隶属度函数编辑器（Mfedit）；

⑤ Edit Rules——打开模糊规则编辑器（Ruleedit）；

⑥ Edit FIS Properties——修改模糊推理系统的特性。

3. 视图（View）菜单

视图菜单的功能包括：

① View Rules——打开模糊规则浏览器（Ruleview）；

② View Surface——打开模糊系统输入输出曲面视图（Surfview）。

5.3.2 隶属度函数编辑器（Mfedit）

在 MATLAB 命令窗口输入：mfedit，或在模糊推理系统编辑器中选择编辑图 5-15 隶属度函数菜单(Edit/Membership Functions)，都可激活隶属度函数编辑器。在该编辑器中，提供了对输入输出语言变量各语言值的隶属度函数类型、参数，进行编辑、修改的图形界面工具，其界面如图 5-16 所示。

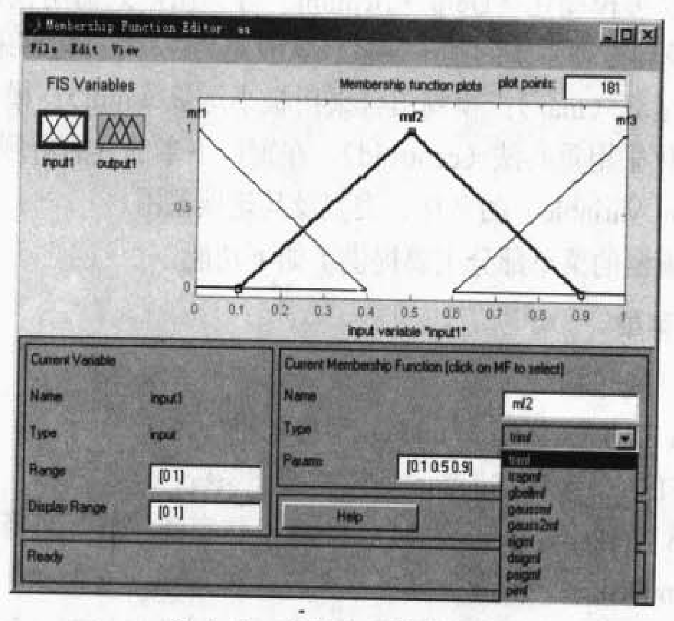


图 5-16　隶属度函数编辑器

在该图形界面中，窗口上半部分为隶属度函数的图形显示，下半部分为隶属度函数的参数设定界面，包括语言变量的名称、论域，隶属度函数的名称、类型和参数。

在菜单部分，文件菜单和视图菜单的功能与模糊推理系统编辑器的文件功能类似。编辑菜单的功能包括添加隶属度函数、添加定制的隶属度函数以及删除隶属度函数等。

5.3.3 模糊规则编辑器（Ruleedit）

在 MATLAB 命令窗口键入：ruleedit，或在模糊推理系统编辑器图 5-15 中选择编辑模糊规则菜单(Edit/Rules)，均可激活模糊规则编辑器。在模糊规则编辑器中，提供了添加、修改和删除模糊规则的图形界面，如图 5-17 所示。

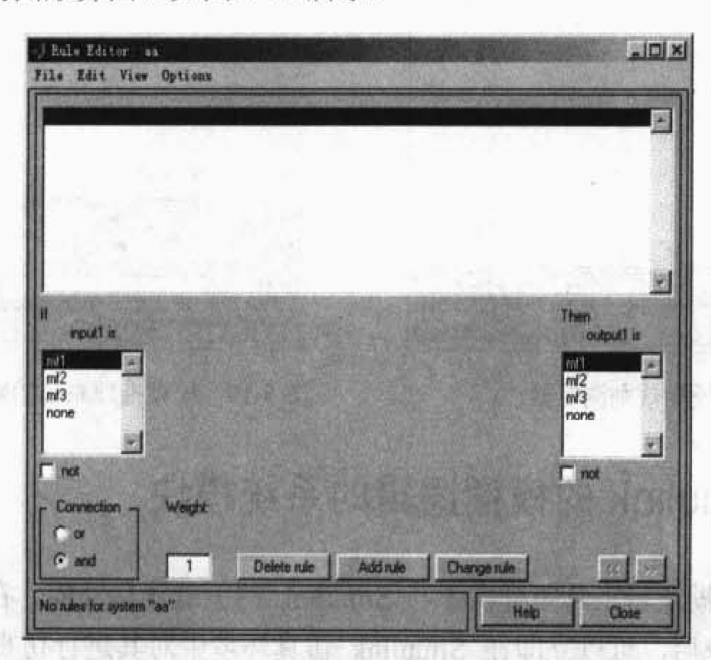


图 5-17 模糊规则编辑器

在模糊规则编辑器图 5-17 中的上部提供了一个文本编辑窗口，用于模糊规则的输入和修改。模糊规则的形式为：IF 条件 THEN 结论。其中条件根据图 5-17 中部左半输入变量窗口中的内容选择。当有两个输入变量时，还需利用图 5-17 左下角 Connection 部分选择两个输入变量间的关系是或（or）还是与（and）。结论根据图 5-17 中部右半输出变量窗口中的内容选择。在窗口的下部有一个下拉列表框，供用户选择某一规则类型。三个按钮【Delete rule】、【Add rule】和【Change rule】分别用于删除、增加和修改模糊规则。

模糊规则编辑器的菜单功能与前两种编辑器基本类似，在其视图菜单中能够激活其他的编辑器或窗口。

5.3.4 模糊规则浏览器（Ruleview）

在 MATLAB 命令窗口输入：ruleview，或在上述三种编辑器中选择相应菜单（View/View Rules），都可激活模糊规则浏览器。在模糊规则浏览器中，以图形形式描述了模糊推理系统的推理过程，其界面如图 5-18 所示。

5.3.5 模糊推理输入输出曲面视图（Surfview）

在 MATLAB 命令窗口键入 surfview 命令，或在各个编辑器窗口选择相应菜单（View/View Surface），即可打开模糊推理的输入输出曲面视图窗口。该窗口以图形的形式显示模糊推理系统的输入输出特性曲面，其界面如图 5-19 所示。



图 5-18 模糊规则浏览器

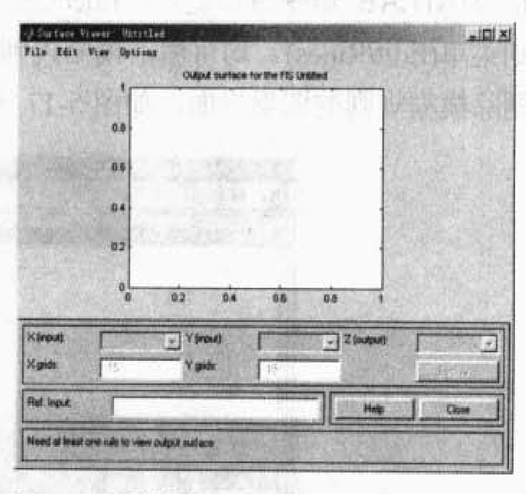


图 5-19 模糊推理系统的输入输出曲面视图

5.4 基于 Simulink 的模糊逻辑的系统模块

MATLAB 的模糊逻辑工具箱提供了与 Simulink 的无缝连接功能。在模糊逻辑工具箱中建立了模糊推理系统后，可以立即在 Simulink 仿真环境中对其进行仿真分析。在 Simulink 中有相应的模糊逻辑控制器方块图（Fuzzy Logic Block），将该方块图复制到用户建立的 Simulink 仿真模型中，并使模糊逻辑控制器方块图的模糊推理矩阵名称与用户在 MATLAB 工作空间（Workspace）建立的模糊推理系统名称相同，即可完成将模糊推理系统与 Simulink 的连接。

Simulink 的模糊逻辑控制器方块图是一个建立在 S-函数 sffis.mex 基础上的屏蔽方块图。该函数的推理算法与模糊逻辑工具箱的 evalfis( )函数相同，但进行了针对 Simulink 仿真应用的优化。

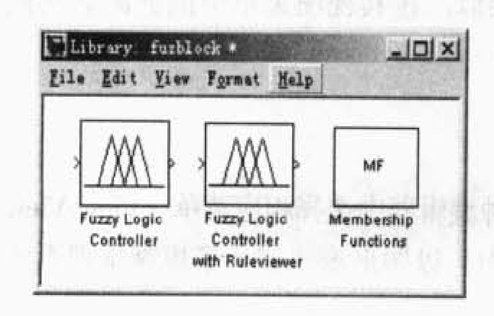


图 5-20 Fuzzy Logic Toolbox 模块库

在 Simulink 库浏览窗口的 Fuzzy Logic Toolbox 节点上，通过单击鼠标右键，便可打开如图 5-20 所示的 Fuzzy Logic Toolbox 模块库。

在 Fuzzy Logic Toolbox 模块库中包含了以下三种模块。

① 模糊逻辑控制器（Fuzzy Logic Controller）;

② 带有规则浏览器的模糊逻辑控制器（Fuzzy

Logic Controller with Ruleviewer)；

③ 隶属度函数模块库（Membership Functions）。

用鼠标双击隶属度函数模块库（Membership Functions）的图标，便可打开如图 5-21 所示的隶属度函数模块库，它包含了多种隶属度函数模块。

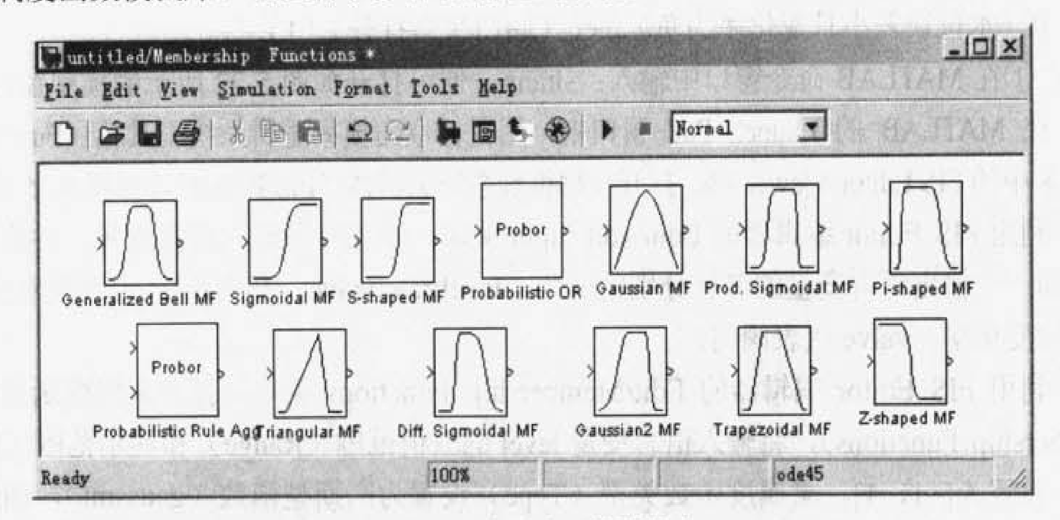


图 5-21　隶属度函数模块库

下面仅以 MATLAB 模糊工具箱中提供的一个水位模糊控制系统仿真的实例，来说明模糊逻辑控制器（Fuzzy Logic Controller）的使用。

例 5-1　一个水位控制系统的 Simulink 仿真模型如图 5-22 所示。

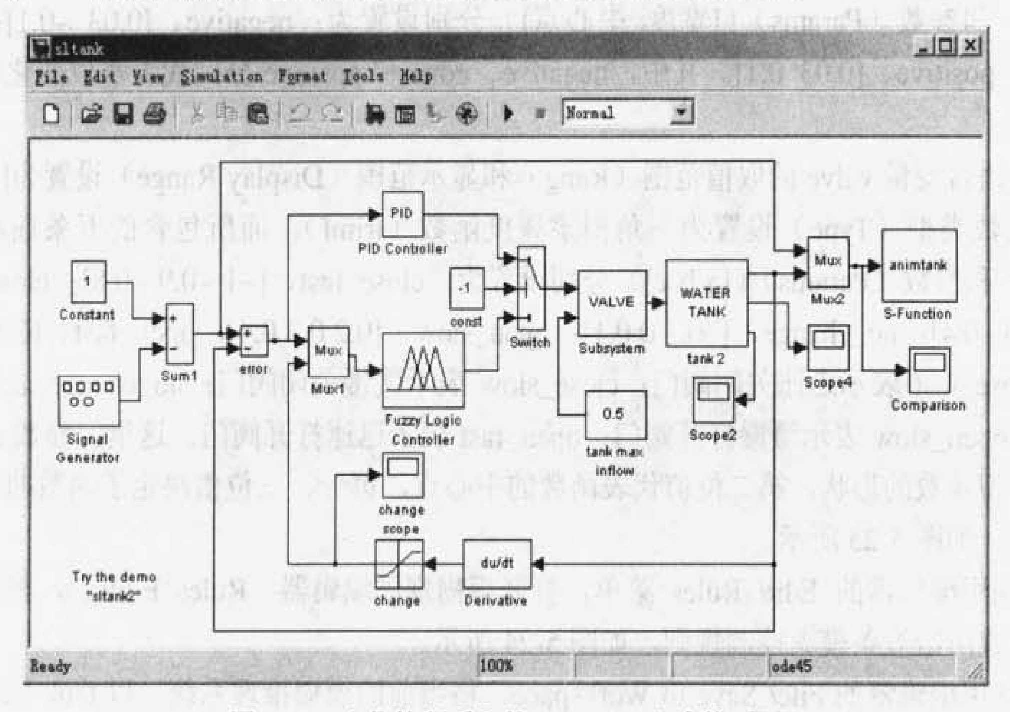


图 5-22　水位控制系统的 Simulink 仿真模型图

采用如下的简单模糊控制规则：

① If（水位误差小）then（阀门大小不变）(1)；

② If（水位低）then（阀门迅速打开）(1)；

③ If（水位高）then（阀门迅速关闭）(1)；

④ If（水位误差小且变化率为正）then（阀门缓慢关闭）(1)；

⑤ If（水位误差小且变化率为负）then（阀门缓慢打开）(1)。

解：①在 MATLAB 命令窗口中输入：sltank，便可打开如图 5-22 所示的模型窗口。

② 在 MATLAB 的 Launch Pad 窗口中，用鼠标双击模糊逻辑系统工具箱（Fuzzy Logic Toolbox）中的 FIS Editor Viewer 项，打开模糊推理系统编辑器（FIS Editor），如图 5-15 所示。

③ 利用 FIS Editor 编辑器的 Edit/Add input 菜单，添加一条输入语言变量，并将两个输入语言和一个输出语言变量的名称分别定义为：level;rate;valve。其中，level 代表水位；rate 代表水位变化率；valve 代表阀门。

④ 利用 FIS Editor 编辑器的 Edit/Membership Functions 菜单，打开隶属度函数编辑器（Membership Functions），将输入语言变量 level 的取值范围（Range）和显示范围（Display Range）设置为[−1，1]，隶属度函数类型（Type）设置为高斯型函数（gaussmf），而所包含的三条曲线的名称（Name）和参数（Params）([宽度　中心点])分别设置为：high、[0.3 −1]；okay、[0.3 0]；low、[0.3 1]。其中，high、okay 和 low 分别代表水位高、刚好和低。

将输入语言变量 rate 取值范围（Range）和显示范围（Display Range）设置为[−0.1，0.1]，隶属度函数类型（Type）设置为高斯型函数（gaussmf），而所包含的三条曲线的名称（Name）和参数（Params）([宽度　中心点])分别设置为：negative、[0.03 −0.1]；none、[0.03 0]；positive、[0.03 0.1]。其中，negative、none 和 positive 分别代表水位变化率为负、不变和正。

输出语言变量 valve 的取值范围（Rang）和显示范围（Display Range）设置为[−1，1]，隶属度函数类型（Type）设置为三角型隶属度函数（trimf），而所包含的五条曲线的名称（Name）和参数（Params）([a b c])分别设置为：close_fast、[−1 −0.9 −0.8]；close_slow、[−0.6 −0.5 −0.4]；no_change、[−0.1 0 0.1]；open_slow、[0.2 0.3 0.4]；open_fast、[0.8 0.9 1]。其中，close_fast 表示迅速关闭阀门；close_slow 表示缓慢关闭阀门；no_change 表示阀门大小不变；open_slow 表示缓慢打开阀门；open_fast 表示迅速打开阀门。这里，参数 a、b 和 c 指定三角型函数的形状，第二位值代表函数的中心点，第一、三位值决定了函数曲线的起始和终止点，如图 5-23 所示。

⑤ 利用编辑器的 Edit/ Rules 菜单，打开模糊规则编辑器（Rules Editor），根据题给规则分别设置所给五条模糊控制规则，如图 5-24 所示。

⑥ 利用编辑器的 File/ Save to Workspace，将当前的模糊推理系统，以 tank 保存到工作空间中。

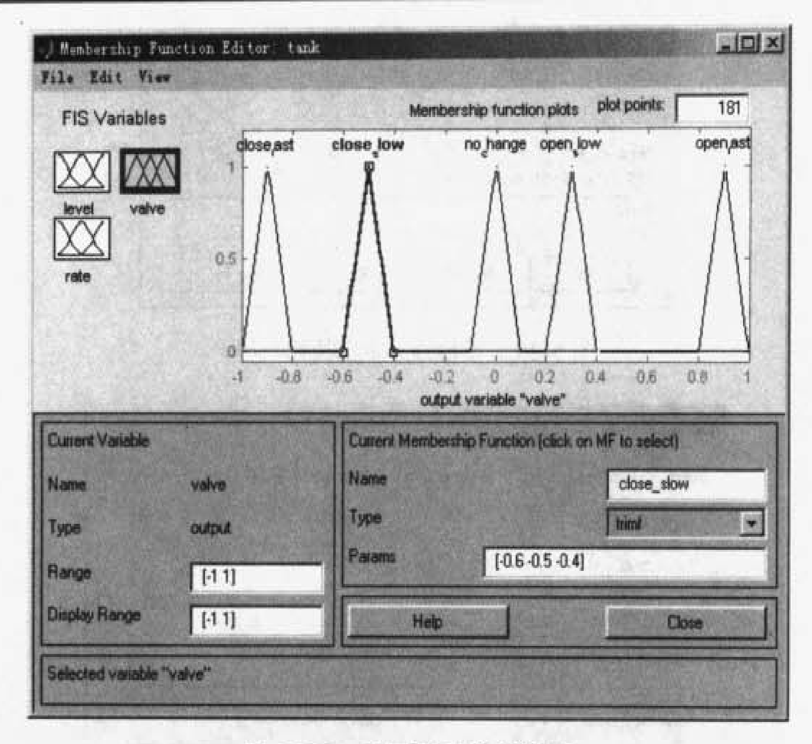


图 5-23　隶属度函数编辑器

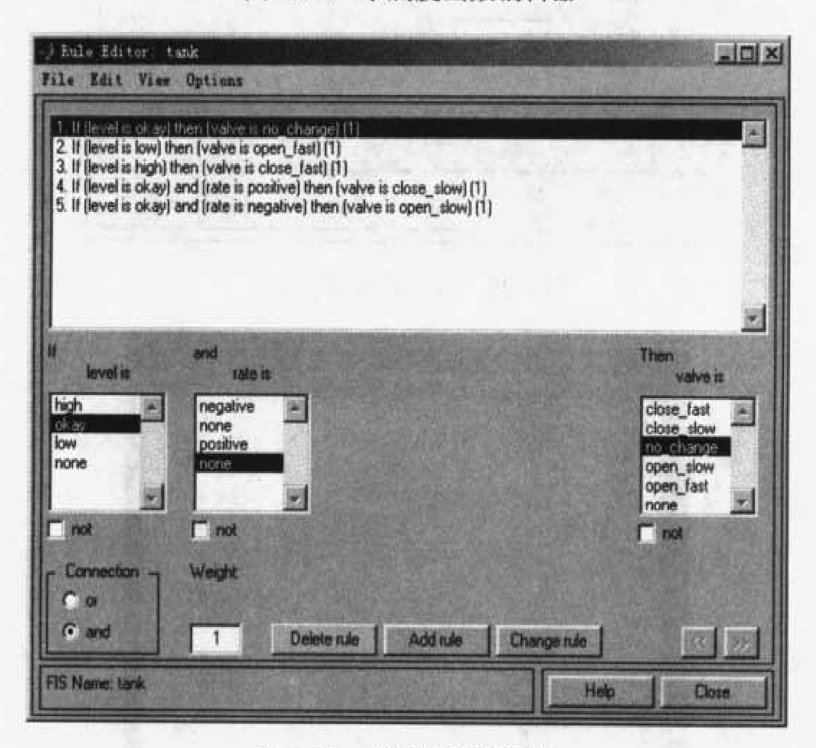


图 5-24　模糊规则编辑器

⑦ 在如图 5-22 所示的 Simulink 仿真系统中，打开 Fuzzy Logic Controller 模糊逻辑控制器模块对话框，在其 FIS File or Structure 参数对话框中输入：tank，如图 5-25 所示。

⑧ 在如图 5-22 所示的 Simulink 系统中，打开如图 5-26 所示的仿真参数设置窗口，正确设置仿真参数后，启动仿真，便可看到如图 5-27 所示的水位变化曲线。

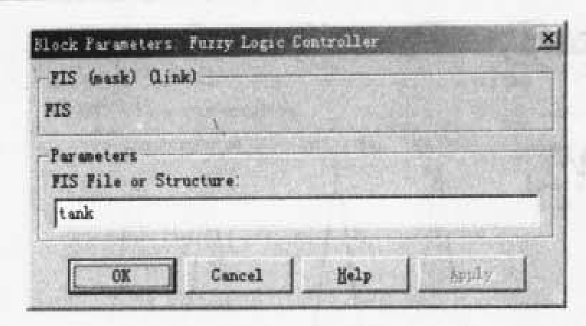


图 5-25　模糊逻辑控制器对话框

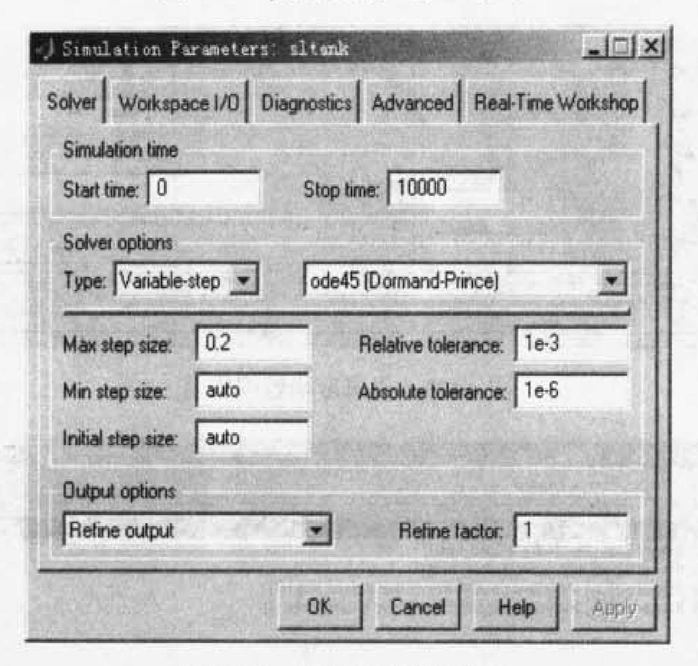


图 5-26　仿真参数设置窗口

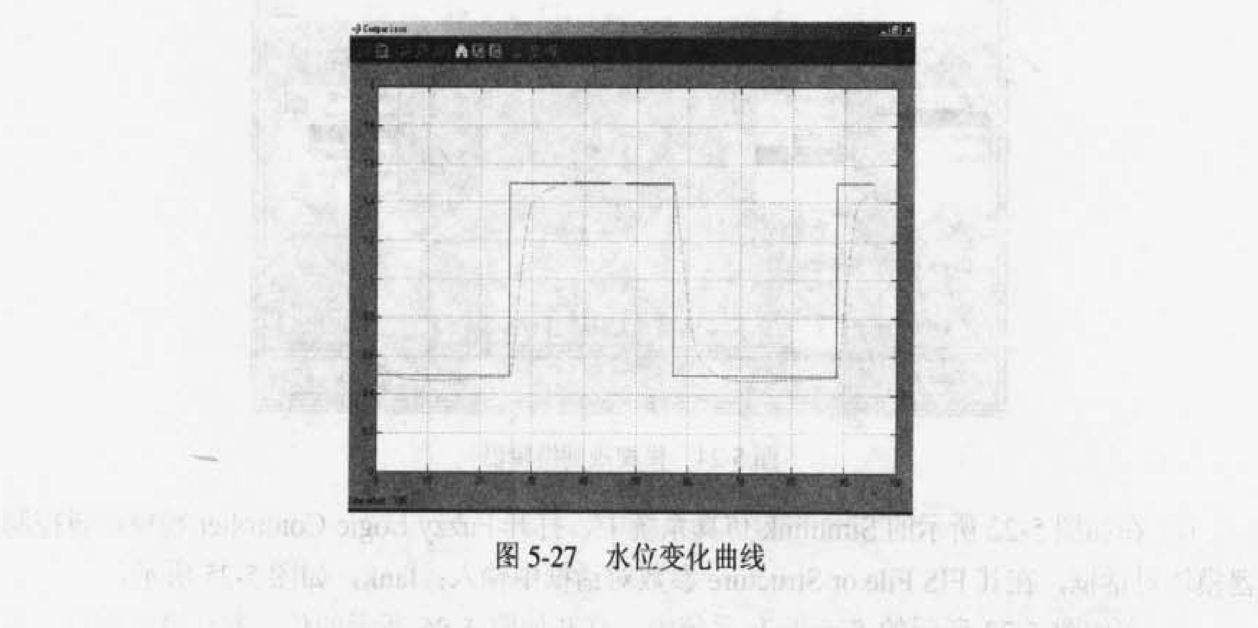


图 5-27　水位变化曲线

第6章　模糊神经和模糊聚类及其MATLAB实现

目前，模糊神经网络控制在控制领域里已经成为一个研究热点，其原因在于两者之间的互补性质。神经网络和模糊系统均属于无模型的估计器和非线性动力学系统，也是一种处理不确定性、非线性和其他不确定问题（ill-posed problem）的有力工具。但两者之间的特性却存在很大的差异。模糊系统中知识的抽取和表达比较方便，它比较适合于表达那些模糊或定性的知识，其推理方式比较类似于人的思维模式。但一般说来，模糊系统缺乏自学习和自适应能力，要设计和实现模糊系统的自适应控制是比较困难的。而神经网络则可直接从样本中进行有效的学习，它具有并行计算、分布式信息存储、容错能力强以及具备自适应学习功能等一系列优点。正是由于这些优点，神经网络的研究受到广泛的关注并吸引了许多研究工作者的兴趣。但一般说来，神经网络不适于表达基于规则的知识，因此在对神经网络进行训练时，由于不能很好地利用已有的经验知识，常常只能将初始权值取为零或随机数，从而增加了网络的训练时间或者陷入非要求的局部极值。总的来说，神经网络适合于处理非结构化信息，而模糊系统对处理结构化的知识更为有效。

基于上述讨论可以想见，若能将模糊逻辑与神经网络适当地结合起来，吸取两者的长处，则可组成比单独的神经网络系统或单独的模糊系统性能更好的系统。

在 MATLAB 模糊逻辑工具箱中，提供了有关模糊逻辑推理的高级应用，包括自适应、模糊聚类和给定数据的模糊建模。下面首先介绍用神经网络来实现模糊系统的两种结构。

6.1　基于标准模型的模糊神经网络

由前面已知，在模糊系统中，模糊模型的表示主要有两种：一种是模糊规则的后件是输出量的某一模糊集合，称它为模糊系统的标准模型或 Mamdani 模型；另一种是模糊规则的后件是输入语言变量的函数，典型的情况是输入变量的线性组合，称它为模糊系统的 Takagi-Sugeno 模型。下面首先讨论基于标准模型的模糊神经网络。

6.1.1　模糊系统的标准模型

在前面已经介绍过，对于多输入多输出（MIMO）的模糊规则可以分解为多个多输入单输出（MISO）的模糊规则。因此，为了不失一般性，下面只讨论 MISO 模糊系统。

图 6-1 为一基于标准模型的 MISO 模糊系统的原理结构图。其中，$x\in \boldsymbol{R}^n$，$y\in \boldsymbol{R}$。如果该模糊系统的输出作用于一个控制对象，那么它的作用便是一个模糊逻辑控制器；否则，它

可用于模糊逻辑决策系统、模糊逻辑诊断系统等其他方面。

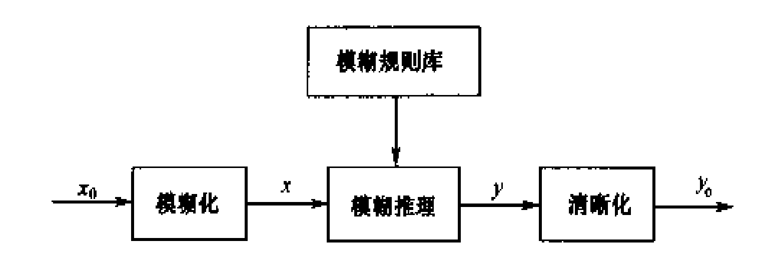


图 6-1 基于标准模型的 MISO 模糊系统原理结构图

设输入向量 $\boldsymbol{x}=[x_1\ x_2\ \cdots\ x_n]^{\mathrm{T}}$，每个分量 x_i 均为模糊语言变量，并设

$$T(x_i)=\{A_i^1, A_i^2, \cdots, A_i^{m_i}\}\quad i=1,2,\cdots,n$$

式中，A_i^j $(j=1, 2, \cdots, m_i)$是 x_i 的第 j 个语言变量值，它是定义在论域 U_i 上的一个模糊集合；相应的隶属度函数为 $\mu_{A_i^j}(x_i)$ $(i=1,2,\cdots,n; j=1,2,\cdots,m_i)$。

输出量 y 也为模糊语言变量，且 $T(y)=\{B^1, B^2, \cdots, B^{m_y}\}$。式中，$B^j$ $(j=1,2,\cdots,m_y)$是 y 的第 j 个语言变量值，它是定义在论域 U_y 上的模糊集合；相应的隶属度函数为 $\mu_{B^j}(y)$。

设描述输入输出关系的模糊规则为

R_i：如果 x_1 是 A_1^i and x_2 是 $A_2^i \cdots$ and x_n 是 A_n^i 则 y 是 B_i

式中，$i=1,2,\cdots,m$, m 表示规则总数，$m \leqslant m_1 m_2 \cdots m_n$。

若输入量采用单点模糊集合的模糊化方法，则对于给定的输入 $\boldsymbol{x}$，可以求得对于每条规则的适用度为

$$\alpha_i=\mu_{A_1^i}(x_1)\wedge\mu_{A_2^i}(x_2)\wedge\cdots\wedge\mu_{A_n^i}(x_n)$$

或

$$\alpha_i=\mu_{A_1^i}(x_1)\mu_{A_2^i}(x_2)\dots\mu_{A_n^i}(x_n)$$

通过模糊推理可得对于每一条模糊规则的输出量模糊集合 B_i 的隶属度函数为

$$\mu_{B_i}(y)=\alpha_i\wedge\mu_{B_i}(y)$$

或

$$\mu_{B_i}(y)=\alpha_i\mu_{B_i}(y)$$

从而输出量总的模糊集合为

$$B=\bigcup_{i=1}^{m}B_i$$

$$\mu_B(y)=\bigvee_{i=1}^{m}\mu_{B_i}(y)$$

若采用加权平均的清晰化方法，则可求得输出的清晰化量为

$$y_0=\frac{\int_{U_y} y\mu_B(y_0)\mathrm{d}y}{\int_{U_y}\mu_B(y_0)\mathrm{d}y}$$

由于计算上式的积分很麻烦，实际计算时通常用下面的近似公式

$$y_0=\frac{\sum_{i=1}^{m} y_{c_i}\mu_{B_i}(y_{c_i})}{\sum_{i=1}^{m}\mu_{B_i}(y_{c_i})}$$

式中，y_{c_i} 是 $\mu_{B_i}(y)$ 取最大值的点，它一般也就是隶属度函数的中心点。显然

$$\mu_{B_i}(y_{c_i})=\max_{y}\mu_{B_i}(y)=\alpha_i$$

从而输出量的清晰化量表达式可变为

$$y_0=\sum_{i=1}^{m} y_{c_i}\bar{\alpha}_i$$

其中

$$\bar{\alpha}_i=\frac{\alpha_i}{\sum_{j=1}^{m}\alpha_j}$$

6.1.2　系统结构

根据上面给出的模糊系统的模糊模型，可设计出如图 6-2 所示的模糊神经网络结构。图中所示为 MIMO 系统，它是上面所讨论的 MISO 情况的简单推广。

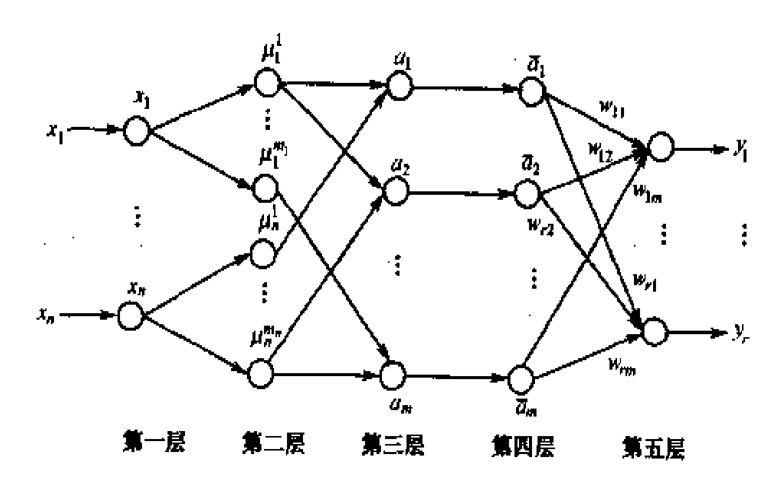


图 6-2　基于标准模型的模糊神经网络结构图

图中第一层为输入层。该层的各个节点直接与输入向量的各分量 x_i 连接，它起着将输入值 $x=[x_1\ x_2\ \cdots\ x_n]^{\mathrm{T}}$ 传送到下一层的作用。该层的节点数 $N_1=n$。

第二层每个节点代表一个语言变量值，如 NB、PS 等。它的作用是计算各输入分量属于各语言变量值模糊集合的隶属度函数μ_i^j，其中

$$\mu_i^j \equiv \mu_{A_i^j}(x_i)$$

$i = 1, 2, \cdots, n, j = 1, 2, \cdots, m_i$。$n$ 是输入量的维数，m_i 是 x_i 的模糊分割数。例如，若隶属函数采用高斯函数表示的铃型函数，则

$$\mu_i^j = \mathrm{e}^{-\frac{(x_i - c_{ij})^2}{\sigma_{ij}^2}}$$

式中，c_{ij} 和 σ_{ij} 分别表示隶属函数的中心和宽度。该层的节点总数 $N_2 = \sum_{i=1}^{n} m_i$。

第三层的每个节点代表一条模糊规则，它的作用是用来匹配模糊规则的前件，计算出每条规则的适用度，即

$$\alpha_j = \min\{\mu_1^{i_1}, \mu_2^{i_2}, \cdots, \mu_n^{i_n}\}$$

或

$$\alpha_j = \mu_1^{i_1}, \mu_2^{i_2}, \cdots, \mu_n^{i_n}$$

式中，$i_1 \in \{1, 2, \cdots, m_1\}, i_2 \in \{1, 2, \cdots, m_2\}, \cdots, i_n \in \{1, 2, \cdots, m_n\}, j = 1, 2, \cdots, m, m = \prod_{i=1}^{n} m_i$。该层的节点总数 $N_3 = m$。对于给定的输入，只有在输入点附近的那些语言变量值才有较大的隶属度值，远离输入点的语言变量值的隶属度或者很小（高斯隶属度函数），或者为 0（三角型隶属度函数）。当隶属度函数很小（如小于 0.05）时，近似取为 0。因此，在α_j 中只有少量节点输出非 0，而多数节点的输出为 0，这一点与前面介绍的局部逼近网络是类似的。

第四层的节点数与第三层相同，即 $N_4 = N_3 = m$，它所实现的是归一化计算，即

$$\bar{\alpha}_j = \frac{\alpha_j}{\sum_{i=1}^{m} \alpha_i}, \quad j = 1, 2, \cdots, m$$

第五层是输出层，它所实现的是清晰化计算，即

$$y_i = \sum_{j=1}^{m} w_{ij} \bar{\alpha}_j, \quad i = 1, 2, \cdots, r$$

与前面所给出的标准模糊模型的清晰化计算相比较，这里的 w_{ij} 相当于 y_i 的第 j 个语言值隶属函数的中心值，上式写成向量形式则为

$$\boldsymbol{y} = \boldsymbol{w}\bar{\boldsymbol{\alpha}}$$

其中

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_r \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{r1} & w_{r2} & \cdots & w_{rm} \end{bmatrix}, \bar{\boldsymbol{\alpha}} = \begin{bmatrix} \bar{\alpha}_1 \\ \bar{\alpha}_2 \\ \vdots \\ \bar{\alpha}_m \end{bmatrix}$$

6.1.3 学习算法

假设各输入分量的模糊分割数是预先确定的，则需要学习的参数主要是最后一层的连接权 $w_{ij}(i=1,2,\cdots,r;j=1,2,\cdots,m)$，第二层的隶属度函数的中心值 c_{ij} 和宽度 $\sigma_{ij}(i=1,2,\cdots,n;j=1,2,\cdots,m_i)$。

上面所给出的模糊神经网络本质上也是一种多层前馈网络，因此，可以仿照 BP 网络用误差反传的方法来设计调整参数的学习算法。为了导出误差反传的迭代算法，需要对每个神经元的输入输出关系加以形式化的描述。

设图 6-3 所示为模糊神经网络中第 q 层第 j 个节点。其中，

节点的纯输入 $=f^{(q)}(x_1^{(q-1)},x_2^{(q-1)},\cdots,x_{n_{q-1}}^{(q-1)};w_{j1}^{(q)},w_{j2}^{(q)},\cdots,w_{jn_{q-1}}^{(q)})$

节点的输出 $=x_j^{(q)}=g^{(q)}(f^{(q)})$。

对于一般的神经元节点，通常有

$$f^{(q)}=\sum_{i=1}^{n_{q-1}}w_{ji}^{(q)}x_i^{(q-1)}$$

$$x_j^{(q)}=g^{(q)}(f^{(q)})=\frac{1}{1+\mathrm{e}^{-\mu f^{(q)}}}$$

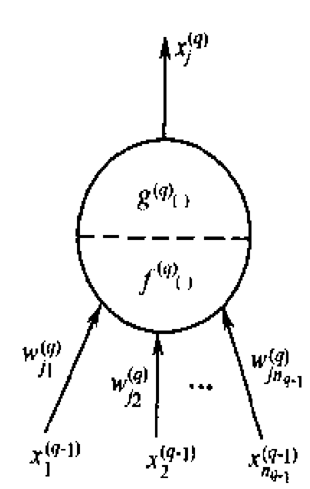


图 6-3　单个神经元节点的基本结构图

而对于图 6-2 所示的模糊神经网络，其神经元节点的输入输出函数则具有较为特殊的形式，下面具体给出它的每一层节点函数。

第一层：$f_i^{(1)}=x_i^{(0)}=x_i\ ,x_i^{(1)}=g_i^{(1)}=f_i^{(1)}\qquad i=1,2,\cdots,n$

第二层：$f_{ij}^{(2)}=-\dfrac{(x_i^{(1)}-c_{ij})^2}{\sigma_{ij}^2}$

$$x_{ij}^{(2)}=\mu_i^j=g_{ij}^{(2)}=\mathrm{e}^{f_{ij}^{(2)}}=\mathrm{e}^{-\frac{(x_i-c_{ij})^2}{\sigma_{ij}^2}}\qquad i=1,2,\cdots,n,\ j=1,2,\cdots,m_i$$

第三层：$f_j^{(3)}=\min\{x_{1i_1}^{(2)},x_{2i_2}^{(2)},\cdots,x_{ni_n}^{(2)}\}=\min\{\mu_1^{i_1},\mu_2^{i_2},\cdots,\mu_n^{i_n}\}$

或者

$$f_j^{(3)}=x_{1i_1}^{(2)}x_{2i_2}^{(2)}\cdots x_{ni_n}^{(2)}=\mu_1^{i_1}\mu_2^{i_2}\cdots\mu_n^{i_n}$$

$$x_j^{(3)}=\alpha_j=g_j^{(3)}=f_j^{(3)},\qquad j=1,2,\cdots,m,\quad m=\prod_{i=1}^{n}m_i$$

第四层：$f_j^{(4)}=x_j^{(3)}\Big/\sum_{i=1}^{m}x_i^{(3)}=\alpha_j\Big/\sum_{i=1}^{m}\alpha_i\qquad x_j^{(4)}=\bar{\alpha}_j=g_j^{(4)}=f_j^{(4)}$

$j=1,2,\cdots,m$

第五层：$f_i^{(5)}=\sum_{j=1}^{m}w_{ij}x_j^{(4)}=\sum_{j=1}^{m}w_{ij}\bar{\alpha}_j$，　$x_i^{(5)}=y_i=g_i^{(5)}=f_i^{(5)}$，　$i=1,2,\cdots,r$

设取误差代价函数为

$$E=\frac{1}{2}\sum_{i=1}^{r}(t_i-y_i)^2$$

式中，t_i和y_i分别表示期望输出和实际输出。下面给出误差反传算法来计算$\frac{\partial E}{\partial w_{ij}}$、$\frac{\partial E}{\partial c_{ij}}$和$\frac{\partial E}{\partial \sigma_{ij}}$，然后利用一阶梯度寻优算法来调节$w_{ij}$、$c_{ij}$和$\sigma_{ij}$。

首先计算

$$\delta_i^{(5)}\equiv-\frac{\partial E}{\partial f_i^{(5)}}=-\frac{\partial E}{\partial y_i}=t_i-y_i$$

进而求得

$$\frac{\partial E}{\partial w_{ij}}=\frac{\partial E}{\partial f_i^{(5)}}\frac{\partial f_i^{(5)}}{\partial w_{ij}}=-\delta_i^{(5)}x_j^{(4)}=-(t_i-y_i)\bar{\alpha}_j$$

再计算

$$\delta_j^{(4)}\equiv-\frac{\partial E}{\partial f_j^{(4)}}=-\sum_{i=1}^{r}\frac{\partial E}{\partial f_i^{(5)}}\frac{\partial f_i^{(5)}}{\partial g_j^{(4)}}\frac{\partial g_j^{(4)}}{\partial f_j^{(4)}}=\sum_{i=1}^{r}\delta_i^{(5)}w_{ij}$$

$$\delta_j^{(3)}\equiv-\frac{\partial E}{\partial f_j^{(3)}}=-\sum_{i=1}^{r}\frac{\partial E}{\partial f_j^{(4)}}\frac{\partial f_j^{(4)}}{\partial g_j^{(3)}}\frac{\partial g_j^{(3)}}{\partial f_j^{(3)}}=\delta_j^{(4)}\sum_{\substack{i=1\\i\neq j}}^{m}x_i^{(3)}\Bigg/\left(\sum_{i=1}^{m}x_i^{(3)}\right)^2=\delta_j^{(4)}\sum_{\substack{i=1\\i\neq j}}^{m}\alpha_i\Bigg/\left(\sum_{i=1}^{m}\alpha_i\right)^2$$

$$\delta_{ij}^{(2)}\equiv-\frac{\partial E}{\partial f_{ij}^{(2)}}=-\sum_{k=1}^{m}\frac{\partial E}{\partial f_k^{(3)}}\frac{\partial f_k^{(3)}}{\partial g_{ij}^{(2)}}\frac{\partial g_{ij}^{(2)}}{\partial f_{ij}^{(2)}}=\sum_{k=1}^{m}\delta_k^{(3)}s_{ij}\mathrm{e}^{f_{ij}^{(2)}}=\sum_{k=1}^{m}\delta_k^{(3)}s_{ij}\mathrm{e}^{-\frac{(x_i-c_{ij})^2}{\sigma_{ij}^2}}$$

当$f^{(3)}$采用取小运算时，则当$g_{ij}^{(2)}=\mu_i^j$是第k个规则节点输入的最小值时

$$s_{ij}=\frac{\partial f_k^{(3)}}{\partial g_{ij}^{(2)}}=\frac{\partial f_k^{(3)}}{\partial \mu_i^j}=1$$

否则

$$s_{ij}=\frac{\partial f_k^{(3)}}{\partial g_{ij}^{(2)}}=\frac{\partial f_k^{(3)}}{\partial \mu_i^j}=0$$

当$f^{(3)}$采用相乘运算时，则当$g_{ij}^{(2)}=\mu_i^j$是第k个规则节点的一个输入时

$$s_{ij}=\frac{\partial f_k^{(3)}}{\partial g_{ij}^{(2)}}=\frac{\partial f_k^{(3)}}{\partial \mu_i^j}=\prod_{\substack{j=1\\ j\neq i}}^{n}\mu_j^{i_j}$$

否则

$$s_{ij}=\frac{\partial f_k^{(3)}}{\partial g_{ij}^{(2)}}=\frac{\partial f_k^{(3)}}{\partial \mu_i^j}=0$$

从而可得所求一阶梯度为

$$\frac{\partial E}{\partial c_{ij}}=\frac{\partial E}{\partial f_{ij}^{(2)}}\frac{\partial f_{ij}^{(2)}}{\partial c_{ij}}=-\delta_{ij}^{(2)}\frac{2(x_i-c_{ij})}{\sigma_{ij}^2}$$

$$\frac{\partial E}{\partial \sigma_{ij}}=\frac{\partial E}{\partial f_{ij}^{(2)}}\frac{\partial f_{ij}^{(2)}}{\partial \sigma_{ij}}=-\delta_{ij}^{(2)}\frac{2(x_i-c_{ij})^2}{\sigma_{ij}^3}$$

在求得所需的一阶梯度后，最后可给出参数调整的学习算法为

$$w_{ij}(k+1)=w_{ij}(k)-\beta\frac{\partial E}{\partial w_{ij}}\qquad i=1,2,\cdots,r\qquad j=1,2,\cdots,m$$

$$c_{ij}(k+1)=c_{ij}(k)-\beta\frac{\partial E}{\partial c_{ij}}\qquad i=1,2,\cdots,n\qquad j=1,2,\cdots,m_i$$

$$\sigma_{ij}(k+1)=\sigma_{ij}(k)-\beta\frac{\partial E}{\partial \sigma_{ij}}\qquad i=1,2,\cdots,n\qquad j=1,2,\cdots,m_i$$

式中，$\beta>0$，为学习率。

该模糊神经网络也和 BP 网络及 RBF 网络等一样，本质上也是实现从输入到输出的非线性映射。它与 BP 网络一样，结构上都是多层前馈网，学习算法都是通过误差反传的方法；它与 RBF 网络等一样，都属于局部逼近网络。

6.2 基于 Takagi-Sugeno 模型的模糊神经网络

在前面对模糊推理的讨论中，分别介绍了 Mamdani 型模糊推理和 Takagi-Sugeno 型模糊推理。MATLAB 模糊逻辑工具箱同时提供对这两种模糊推理方法的支持。其中对 Takagi-Sugeno 型模糊推理仅支持一阶规则，即规则的输出为输入变量的线性组合。

Mamdani 型模糊推理和 Takagi-Sugeno 型模糊推理各有优缺点。对 Mamdani 型模糊推理，由于其规则的形式符合人们思维和语言表达的习惯，因而能够方便地表达人类的知识，但存在计算复杂、不利于数学分析的缺点。而 Takagi-Sugeno 型模糊推理则具有计算简单，利于数学分析的优点，且易于和 PID 控制方法以及优化、自适应方法结合，从而实现具有优化与自适应能力的控制器或模糊建模工具。

根据 Takagi-Sugeno 型模糊推理的特点，有关学者将其与神经网络结合，用于构造具有

自适应学习能力的神经模糊系统。模糊逻辑与神经网络的结合，是近年来计算智能学科的一个重要研究方向。两者结合形成的模糊神经网络，同时具有模糊逻辑易于表达人类知识和神经网络的分布式信息存储以及学习能力的优点，对于复杂系统的建模和控制提供了有效的工具。

6.2.1 模糊系统的 Takagi-Sugeno 模型

由于 MIMO 的模糊规则可分解为多个 MISO 模糊规则，因此下面也只讨论 MISO 模糊系统的模型。

设输入向量 $\boldsymbol{x}=[x_1\ x_2\ \cdots\ x_n]^{\mathrm{T}}$，每个分量 x_i 均为模糊语言变量，并设

$$T(x_i)=\{A_i^1, A_i^2, \cdots, A_i^{m_i}\} \quad i=1,2,\cdots,n$$

式中，A_i^j ($j=1,2,\cdots,m_i$)是 x_i 的第 j 个语言变量值，它是定义在论域 U_i 上的一个模糊集合。相应的隶属度函数为 $\mu_{A_i^j}(x_i)$($i=1,2,\cdots,n; j=1,2,\cdots,m_i$)。

Takagi 和 Sugeno 所提出的模糊规则后件是输入变量的线性组合，即

$\boldsymbol{R}_j$: 如果 x_1 是 A_1^j and x_2 是 A_2^j and…and x_n 是 A_n^j，则

$$y_j=p_{j0}+p_{j1}x_1+\cdots+p_{jn}x_n$$

式中，$j=1,2,\cdots,m, m\leqslant\prod_{i=1}^{n}m_i$。

若输入量采用单点模糊集合的模糊化方法，则对于给定的输入 $\boldsymbol{x}$，可以求得对于每条规则的适应度为

$$\alpha_j=\mu_{A_1^j}(x_1)\wedge\mu_{A_2^j}(x_2)\wedge\cdots\wedge\mu_{A_n^j}(x_n)$$

或

$$\alpha_j=\mu_{A_1^j}(x_1)\mu_{A_2^j}(x_2)\ldots\ \mu_{A_n^j}(x_n)$$

模糊系统的输出量为每条规则的输出量的加权平均，即

$$y=\frac{\sum_{j=1}^{m}\alpha_j y_j}{\sum_{j=1}^{m}\alpha_j}=\sum_{j=1}^{m}\overline{\alpha}_j y_j$$

其中 $\overline{\alpha}_j=\dfrac{\alpha_j}{\sum_{i=1}^{m}\alpha_i}$

6.2.2 系统结构

根据上面给出的模糊模型，可以设计出如图 6-4 所示的模糊神经网络结构。图中所示为 MIMO 系统，它是上面讨论的 MISO 系统的简单推广。

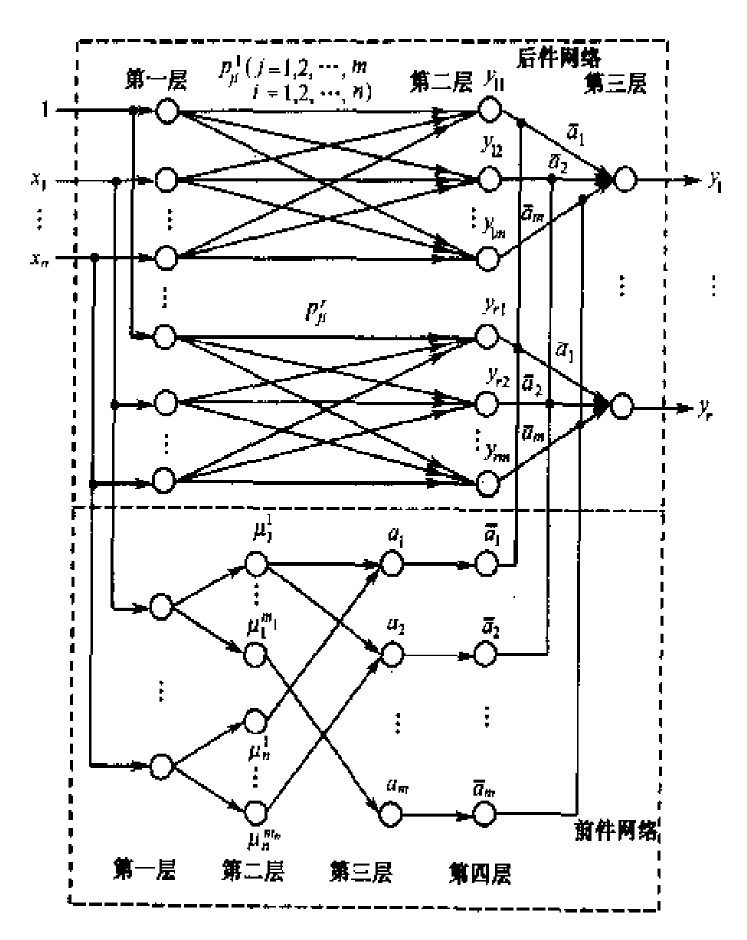


图 6-4　基于 Takagi-Sugeno 模型的模糊神经网络结构图

由图可见,该网络由前件网络和后件网络两部分组成，前件网络用来匹配模糊规则的前件，后件网络用来产生模糊规则的后件。

1．前件网络

前件网络由 4 层组成。第一层为输入层。它的每个节点直接与输入向量的各分量 x_i 连接，它起着将输入值 $\boldsymbol{x}=[x_1\ x_2\ \cdots\ x_n]^{\mathrm{T}}$ 传送到下一层的作用。该层的节点数 $N_1=n$。

第二层每个节点代表一个语言变量值，如 NM、PS 等。它的作用是计算各输入分量属于各语言变量值模糊集合的隶属度后函数 μ_i^j，即

$$\mu_i^j=\mu_{A_i^j}(x_i)$$

式中，$i=1,2,\cdots,n,\ j=1,2,\cdots,m_i$；$n$ 是输入量的维数；m_i 是 x_i 的模糊分割数。例如，若隶属函数采用高斯函数表示的铃型函数，则

$$\mu_i^j=\mathrm{e}^{-\frac{(x_i-c_{ij})^2}{\sigma_{ij}^2}}$$

式中，c_{ij} 和 σ_{ij} 分别表示隶属函数的中心和宽度。该层的节点总数 $N_2=\sum\limits_{i=1}^{n}m_i$。

第三层的每个节点代表一条模糊规则，它的作用是用来匹配模糊规则的前件，计算出每条规则的适应度，即

$$\alpha_j = \min\{\mu_1^{i_1}, \mu_2^{i_2}, \cdots, \mu_n^{i_n}\}$$

或

$$\alpha_j = \mu_1^{i_1}\mu_2^{i_2}\cdots\mu_n^{i_n}$$

式中，$i_1 \in \{1,2,\cdots,m_1\}, i_2 \in \{1,2,\cdots,m_2\},\cdots,i_n \in \{1,2,\cdots,m_n\}, j=1,2,\cdots,m, m=\prod_{i=1}^{n} m_i$。

该层的节点总数 $N_3=m$。对于给定的输入，只有在输入点附近的语言变量值才有较大的隶属度值，远离输入点的语言变量值的隶属度或者很小（高斯型隶属度函数），或者为 0（三角型隶属度函数）。当隶属度函数很小（如小于 0.05）时，近似取为 0。因此，在α_j中只有少量节点输出非 0，而多数节点的输出为 0，这一点类似于局部逼近网络。

第四层的节点数与第三层相同，$N_4=N_3=m$，它所实现的是归一化计算，即

$$\overline{\alpha}_j = \frac{\alpha_j}{\sum_{i=1}^{m}\alpha_i} \qquad i=1,2,\cdots,m$$

2. 后件网络

后件网络由 r 个结构相同的并列子网络所组成，每个子网络产生一个输出量。

子网络的第一层是输入层，它将输入变量传送到第二层。输入层中第 0 个节点的输入值 $x_0=1$，它的作用是提供模糊规则后件中的常数项。

子网络的第二层共有 m 个节点，每个节点代表一条规则，该层的作用是计算每一条规则的后件，即

$$y_{ij} = p_{j0}^i + p_{j1}^i x_1 + \cdots + p_{jn}^i x_n = \sum_{k=0}^{n} p_{jk}^i x_k \qquad j=1,2,\cdots,m, \qquad i=1,2,\cdots,r$$

子网络的第三层是计算系统的输出，即

$$y_i = \sum_{j=1}^{m} \overline{\alpha}_j y_{ij} \qquad i=1,2,\cdots,r$$

可见，y_i 是各规则后件的加权和，加权系数为各模糊规则的经归一化的使用度，即前件网络的输出用做后件网络第三层的连接权值。

至此，图 6-4 所示的神经网络完全实现了 Takagi-Sugeno 的模糊系统模型。

6.2.3 学习算法

假设各输入分量的模糊分割数是预先确定的，那么需要学习的参数主要是后件网络的连接权 p_{ji}^k（$j=1,2,\cdots,m; i=0,1,\cdots,n; k=1,2,\cdots,r$），前件网络第二层各节点隶属函数的中心

值c_{ij}和宽度$\sigma_{ij}(i=1,2,\cdots,m;j=1,2,\cdots,m_i)$。

设取误差代价函数为

$$E=\frac{1}{2}\sum_{i=1}^{r}(t_i-y_i)^2$$

式中，t_i和y_i分别表示期望输出和实际输出。下面首先给出参数p_{ji}^k的学习算法

$$\frac{\partial E}{\partial p_{ji}^k}=\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial y_{kj}}\frac{\partial y_{kj}}{\partial p_{ji}^k}=-(t_k-y_k)\bar{\alpha}_j x_i$$

$$p_{ji}^k(l+1)=p_{ji}^k(l)-\beta\frac{\partial E}{\partial p_{ji}^k}=p_{ji}^k(l)+\beta(t_k-y_k)\bar{\alpha}_j x_i$$

式中，$j=1,2,\cdots,m;i=0,1,\cdots,n;k=1,2,\cdots,r$。

下面讨论c_{ij}及σ_{ij}的学习问题，这时可将参数p_{ji}^k固定。图 6-4 可以简化为如图 6-5 所示。这时每条规则的后件在简化结构中变成了最后一层的连接权。

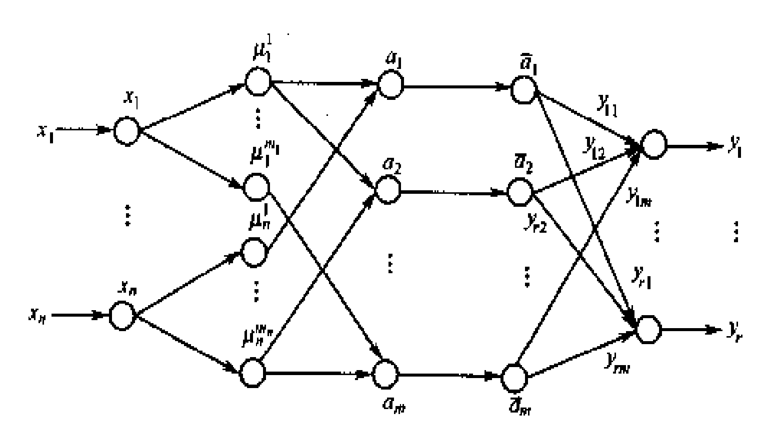


图 6-5 基于 Takagi-Sugeno 模型的模糊神经网络简化结构图

比较图 6-5 与图 6-2 可以发现，该简化结构与基于标准模型的模糊神经网络具有完全相同的结构，这时只需令最后一层的连接权$y_{ij}=w_{ij}$，则完全可以借用前面已得的结果，即

$$\delta_i^{(5)}=t_i-y_i \qquad i=1,2,\cdots,n$$

$$\delta_j^{(4)}=\sum_{i=1}^{r}\delta_i^{(5)}y_{ij} \qquad j=1,2,\cdots,m$$

$$\delta_j^{(3)}=\delta_j^{(4)}\sum_{\substack{i=1\\i\neq j}}^{m}\alpha_i\bigg/\left(\sum_{i=1}^{m}\alpha_i\right)^2 \qquad j=1,2,\cdots,m$$

$$\delta_{ij}^{(2)}=\sum_{k=1}^{m}\delta_k^{(3)}s_{ij}\mathrm{e}^{-\frac{(x_i-c_{ij})^2}{\sigma_{ij}^2}} \qquad i=1,2,\cdots,n \qquad j=1,2,\cdots,m$$

当 and 采用取小运算时，则当μ_i^j是第k个规则节点输入的最小值时

$$s_{ij}=1$$

否则

$$s_{ij}=0$$

当 and 采用相乘运算时，则当 μ_i^j 是第 k 个规则节点的一个输入时

$$s_{ij}=\prod_{\substack{j=1\\j\neq i}}^{n}\mu_j^{i_j}$$

否则

$$s_{ij}=0$$

最后求得

$$\frac{\partial E}{\partial c_{ij}}=-\delta_{ij}^{(2)}\frac{2(x_i-c_{ij})}{\sigma_{ij}^2}$$

$$\frac{\partial E}{\partial \sigma_{ij}}=-\delta_{ij}^{(2)}\frac{2(x_i-c_{ij})^2}{\sigma_{ij}^3}$$

$$c_{ij}(k+1)=c_{ij}(k)-\beta\frac{\partial E}{\partial c_{ij}}$$

$$\sigma_{ij}(k+1)=\sigma_{ij}(k)-\beta\frac{\partial E}{\partial \sigma_{ij}}$$

式中，β>0，为学习率；$i=1,2,\cdots,n; j=1,2,\cdots,m_i$。

对于上面介绍的两种模糊神经网络，当给定一个输入时，网络（或前件网络）第三层的 $\boldsymbol{\alpha}=[\alpha_1\ \alpha_2\cdots\ \alpha_m]^{\mathrm{T}}$ 中只有少量元素非 0，其余大部分元素均为 0。因此，从 $\boldsymbol{x}$ 到 $\boldsymbol{\alpha}$ 的映射与 RBF 神经网络的非线性映射非常类似。该模糊神经网络也是局部逼近网络。其中第二层的隶属度函数类似于基函数。

模糊神经网络虽然也是局部逼近网络，但是它是按照模糊系统模型建立的，网络中的各个节点及所有参数均有明显的物理意义，因此这些参数的初值可以根据系统的模糊或定性的知识来加以确定，然后利用上述的学习算法可以很快收敛到要求的输入输出关系，这是模糊神经网络比前面单纯的神经网络的优点所在。同时，由于它具有神经网络的结构，因此参数的学习和调整比较容易，这是它与单纯的模糊逻辑系统相比的优点。

基于 Takagi-Sugeno 模型的模糊神经网络可以从另一角度来认识它的输入输出映射关系，若各输入分量的分割是精确的，即相当于隶属度函数为互相拼接的超矩形函数，则网络的输出相当于是原光滑函数的分段线性近似，即相当于用许多块超平面来拟合一个光滑曲面。网络中的 p_{ij}^k 参数便是这些超平面方程的参数，只有当分割越精细时，拟合才能越准确。而实际上这里的模糊分割互相之间是有重叠的，因此即使模糊分割数不多，也能获得光

滑和准确的曲面拟合。基于上面的理解，可以帮助选取网络参数的初值。例如，若根据样本数据或根据其他先验知识已知输出曲面的大致形状时，可根据这些形状来进行模糊分割。若某些部分曲面较平缓，则相应部分的模糊分割可粗些；反之，若某些部分曲面变化剧烈，则相应部分的模糊分割需要精细些。在各分量的模糊分割确定后，可根据各分割子区域所对应的曲面形状用一个超平面来近似，这些超平面方程的参数即作为 p_{ij}^k 的初值。由于网络还要根据给定样本数据进行学习和训练，因此初值参数的选择并不要求很精确。但根据上述的先验知识所作的初步选择却是非常重要的，它可避免陷入不希望的局部极值并大大提高收敛的速度，这一点对于实时控制是尤为重要的。

6.3 MATLAB 模糊神经工具箱函数

在 MATLAB 模糊逻辑工具箱中，提供了有关对模糊神经系统建模和初始化模糊推理系统的函数，见表 6-1。

表 6-1 模糊神经系统函数

函 数 名	功 能
anfis()	模糊神经系统的建模函数
genfis1()	采用网格分割方式生成模糊推理系统函数

6.3.1 模糊神经系统的建模函数

在 MATLAB 模糊逻辑工具箱中，提供了对基于 Takagi-Sugeno 型模糊推理的模糊神经系统建模的方法，该模糊推理系统利用反向传播算法和最小二乘算法来完成对输入/输出数据对的建模。相应的函数为 anfis()，该函数的输出为一个三维或五维向量。当未指定检验数据时，输出向量为三维。anfis()支持采用输出加权平均的一阶 Takagi-Sugeno 型模糊推理；该函数的调用格式为

```
[fisMat1,error1,stepsize]=anfis (trnData)
[fisMat1,error1,stepsize]=anfis (trnData,fisMat)
[fisMat1,error1,stepsize]=anfis(trnData,fisMat,trnOpt,dispOpt)
[fisMat1,error1,stepsize,fisMat2,error2]=anfis(trnData, trnOpt,dispOpt,chkData)
[fisMat1,error1,stepsize,fisMat2,error2]=anfis(trnData, fisMat,trnOpt,dispOpt,chkData)
```

式中，trnData 为训练学习的输入输出数据矩阵，该矩阵的每一行对应一组输入输出数据，其最后一列为输出数据；fisMat 是指定初始的模糊推理系统参数（包括隶属度函数类型和参数）的矩阵，该矩阵可以使用函数 fuzzy 通过模糊推理系统编辑器生成，也可使用函数 genfis1()由训练数据直接生成，函数 genfis1()的功能采用网格分割法生成模糊推理系统，其使用方法参见下文的说明；trnOpt 和 dispOpt 分别指定训练的有关选项和在训练执行过程中

MATLAB 命令窗口的显示选项，参数 trnOpt 为一个五维向量，其各个分量的定义如下：trnOpt(1)为训练的次数，默认值为 10；trnOpt(2)为期望误差，默认值为 0；trnOPt(3)为初始步长，默认值为 0.01；trnOpt(4)为步长递减速率，默认值为 0.9；trnOPt(5)为步长递增速率，默认值为 1.1。若 trnOpt 的任一个分量为 NaN（非数值：IEEE 的标准缩写）或被省略，则训练采用默认参数。学习训练的过程在训练参数得到指定值或训练误差得到期望误差时停止。训练过程中的步长调整采用如下的策略：当误差连续四次减小时，则增加步长；当误差变化连续两次出现振荡，即一次增加和一次减少交替发生时，则减小步长。trnOpt 的第四和第五个参数分别按照上述策略控制训练步长的调整；参数 dispOpt 用于控制训练过程中 MATLAB 命令窗口的显示内容，共有四个参数，分别定义如下：dispOPt(1)为显示 ANFIS 的信息，默认值为 1；dispOPt(2)为显示误差测量，默认值为 1；dispOPt(3)为显示训练步长，默认值为 1；dispOPt(4)为显示最终结果，默认值为 1。当 dispOpt 的一个分量为 0 时，则不显示相应内容；若为 1 或为 NaN 或省略，则显示相应内容；chkData 参数为一个与训练数据矩阵有相同列数的矩阵，用于提供检验数据，当提供检验数据时，anfis()返回对检验数据具有最小均方根误差的模糊推理系统 fisMat2；参数 fisMat1 为学习完成后得到的对应最小均方根误差的模糊推理系统矩阵；error1 为训练的均方根误差向量；stepsize 为训练步长向量。在指定检验数据后，输出向量为五维参数向量；参数 fisMat2 为对检验数据具有最小均方根误差的模糊推理系统矩阵；error2 为检验数据对应的最小均方根误差向量。

例 6-1 利用以下 MATLAB 程序可得如图 6-6 所示的输出曲线。

```
x=0:0.1:10;y=sin(2*x)./exp(x/5);
trnData=[x' y'];numMFs=5;
mfType='gbellmf';epoch_n=20;
in_fisMat=genfis1(trnData,numMFs,mfType);
out_fisMat=anfis(trnData,in_fisMat,20);
plot(x,y,'-',x,evalfis(x,out_fisMat),':');
legend('Training Data', 'ANFIs Output')
```

例 6-2 利用以下 MATLAB 程序，设计模糊推理系统。

解：（1）提供训练数据和检验数据的 MATLAB 程序如下：

```
%数据点个数为 51
numPts=51;
x1=linspace(0,1,numPts);
y=.6*sin(pi*x1)+.3*sin(3*pi*x1)+.1*sin(5*pi*x1);
data=[x1' y'];                    %整个数据集
trnData=data(1:2:numPts,:);       %训练数据集
chkData= data(2:2:numPts,:);      %检验数据集
%训练数据和检验数据的分布曲线，如图 6-7 所示。
```

```
plot(trnData(:,1),trnData(:,2),'o',chkData(:,1),chkData(:,2),'x')
```

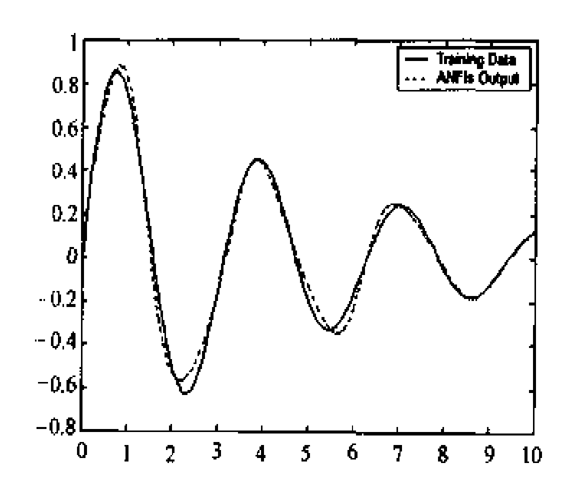


图6-6 训练数据和anfis输出曲线

图6-7 训练数据和检验数据

在这个例子中，不但提供了训练数据，而且提供了检验数据，两种数据在输入空间均匀采样，如图6-7所示。

（2）建立用于模糊建模的Takagi-Sugeno型模糊推理系统的MATLAB程序如下：

```
%采用genfisl()函数直接由训练数据生成模糊推理系统
numMFs=5; %隶属度函数个数
mfType='gbellmf'; %隶属度函数类型
fisMat=genfis1(trnData,numMFs,mfType);
%绘制模糊推理系统的初始隶属度函数，如图6-8所示。
[x,mf]=plotmf(fisMat,'input',1);
plot(x,mf);
title('Initial Membership Functions')
```

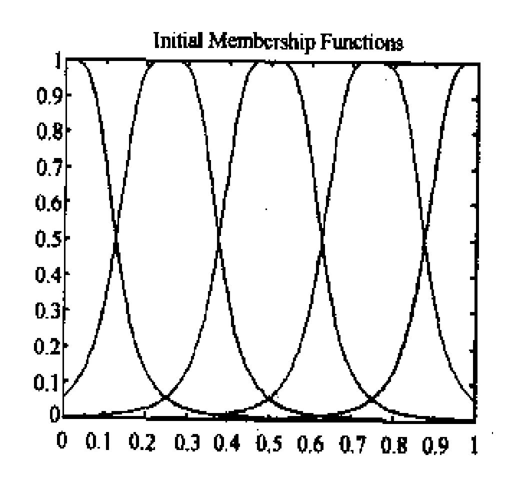


图6-8 genfis()的隶属度函数曲线

如图6-8所示，由函数genfis1( )根据训练数据生成的模糊推理系统隶属度函数曲线。从曲线可以看出，函数genfis1( )按照均匀覆盖输入空间的原则构造了初始隶属度函数。

（3）使用函数anfis( )进行给定数据的神经模糊建模的MATLAB程序如下：

```
numEpochs=40; %训练次数为40
[fisMat1,truErr,ss,fisMat2,chkErr]=anfis(trnData,fisMat,numEpochs,NaN,chkData);
%计算训练后神经模糊系统的输出与训练数据的均方根误差
trnOut=evalfis(trnData(:,1),fisMat1);
trnRMSE=norm(trnOut-trnData(:,2))/sqrt(length(trnOut));
%绘制训练过程中均方根误差的变化情况，如图6-9所示。
epoch=1:numEpochs;
plot(epoch,truErr,'o',epoch,chkErr,'x')
hold on;plot(epoch,[truErr,chkErr]);hold off
%绘制训练过程中步长的变化情况，如图6-10所示。
```

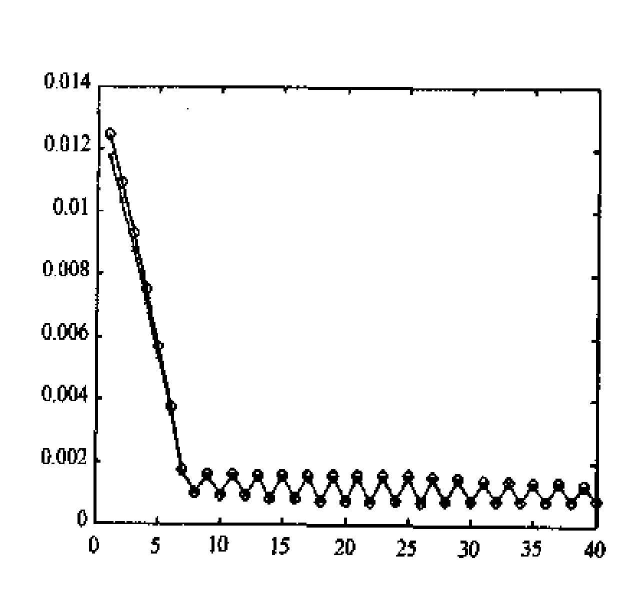


图6-9 训练过程中均方根误差的变化曲线

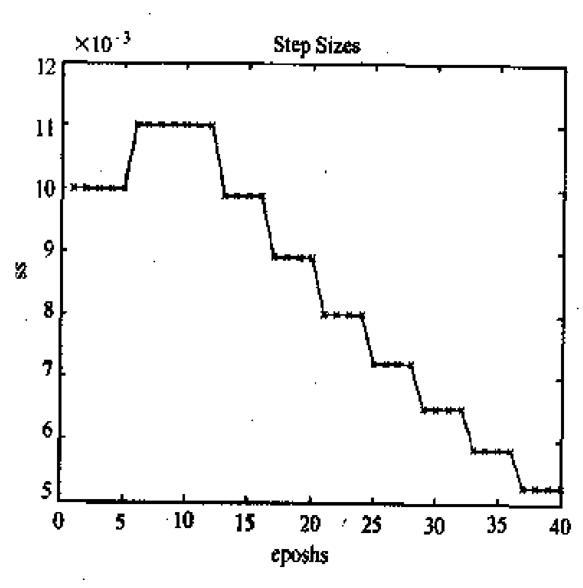


图6-10 训练步长变化曲线

```
plot(epoch,ss,'-',epoch,ss,'x');
xlabel('epochs'); ylabel('ss'); title('Step Sizes');
%绘制训练后模糊推理系统的隶属度函数曲线，如图6-11所示。
[x,mf]=plotmf(fisMat1,'input',1);plot(x,mf)
title('Fiual Membership Functions');
```

从图6-11中可以看出，经过学习后的模糊推理系统提取了训练数据的局部特征。

```
%绘制神经模糊推理系统的输出曲线，如图6-12所示。
anfis_y=evalfis(x1, fisMat1);plot(x1,y, '-', x1,anfis_y, 'x');
```

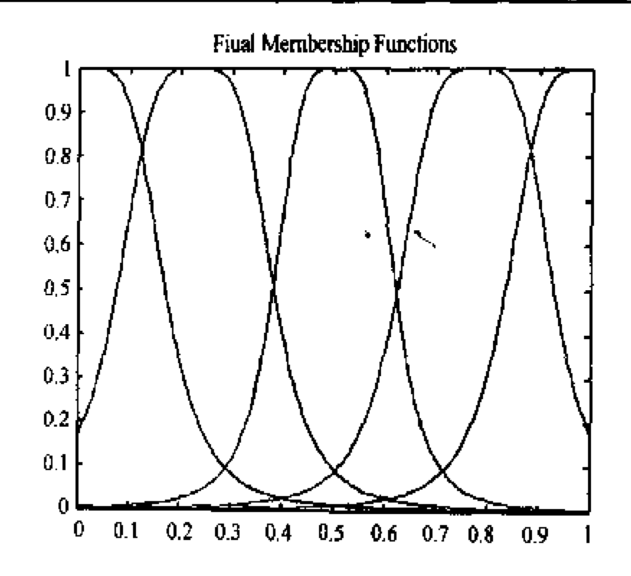


图 6-11　经训练后的模糊推理系统的隶属度函数

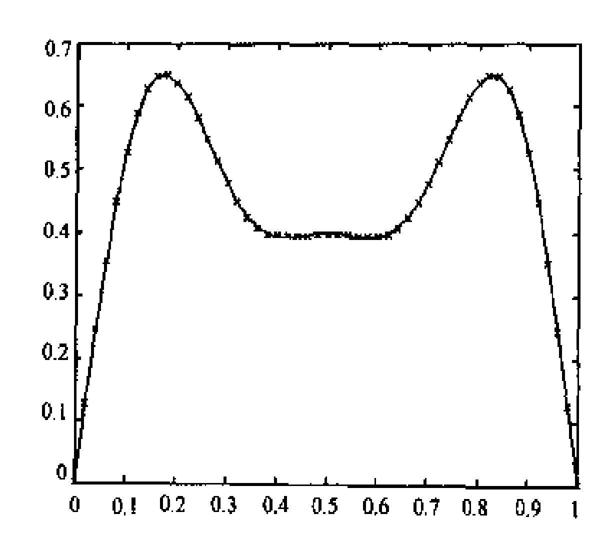


图 6-12　模糊推理系统的输出曲线

6.3.2 采用网格分割方式生成模糊推理系统函数

在给定输入/输出数据后，要利用 anfis()函数进行神经模糊系统建模，必须提供一个以输入/输出数据为基础的初始模糊推理系统。函数 genfis1( )采用网格分割的方式根据给定数据集生成一个模糊推理系统，因而可以与函数 anfis( )配合使用。由 genfis1( )生成的模糊推理系统的输入和隶属度函数的类型及数目可以在使用时指定，也可以采用默认值。函数 genfis1( )的调用格式为

fisMat=genfis1(data)

fisMat=genfis1(data,numMFs,mfType)

式中，data 为给定的输入输出数据集合；numMFs 为一个整数向量，用于指定输入、输出语言变量的隶属度函数个数；参数 mfType 用于指定隶属度函数的类型；fisMat 为生成的模糊推理系统矩阵。当仅使用一个输入参数而不指定隶属度函数个数和类型时，将使用默认值，即隶属度函数个数为 2，类型为钟型曲线，如利用函数 genfis1( )编写的 MATLAB 程序如下：

```
data=[rand(10,1) 10*rand(10,1)-5 rand(10,1)];
numMFs=[3 7];
mfType=str2mat('pimf','trimf');
fisMat=genfis1(data,numMFs,mfType);
[x,mf]=plotmf(fisMat,'input',1);
subplot(2,1,1),plot(x,mf);
xlabel('input 1 (pimf)');
[x,mf]=plotmf(fisMat,'input',2);
subplot(2,1,2),plot(x,mf);
xlabel('input 2 (trimf)');
```



根据以上 MATLAB 程序利用 genfis1( )函数生成的隶属度函数曲线如图 6-13 所示。

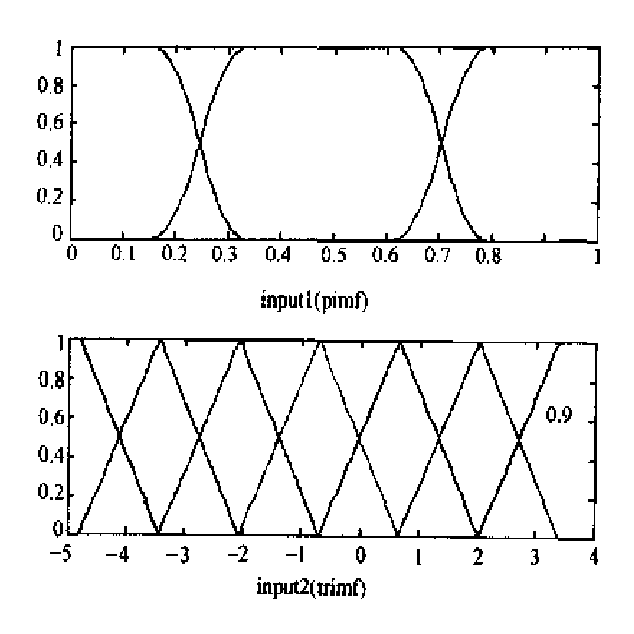


图 6-13 genfis1 生成的隶属度函数曲线

6.3.3 MATLAB 模糊神经推理系统的图形用户界面

为了进一步方便用户，在模糊逻辑工具箱中提供了建立模糊神经推理系统的图形界面工具，该工具以交互式图形界面的形式集成了建立、训练和测试神经模糊推理系统等各种功能。要启动该工具，只需在 MATLAB 命令窗口中键入 anfisedit，即可得到如图 6-14 所示的图形窗口界面。

该图形界面包括的功能主要有：加载数据（Load Data）、生成模糊推理系统（Generate FIS）、训练神经模糊推理系统（Train FIS）和测试神经模糊推理系统（Test FIS）。

1. 加载数据

通过界面上的检查框可以选择加载数据的类型，如训练数据、测试数据和演示数据等。为方便说明，这里选择加载演示数据。加载了演示数据的图形窗口如图 6-15 所示，在图 6-15 的上部显示了数据的变化情况。

2. 生成模糊推理系统

在生成模糊推理系统时，也可通过检查框选择模糊推理系统的生成方法，如网格分割法（Grid partition）、减法聚类方法（Sub. clustering）等。当用鼠标单击生成模糊推理系统的功能按钮时，系统会弹出一个对话框，要求指定模糊推理系统的有关信息，如图 6-16 所示。

在如图 6-16 所示的对话框中，要求输入的信息包括输入语言变量隶属度函数的数目、类型和输出隶属度函数的类型等。选择【OK】按钮后，自动生成一个以 anfis 命名的网络结构。

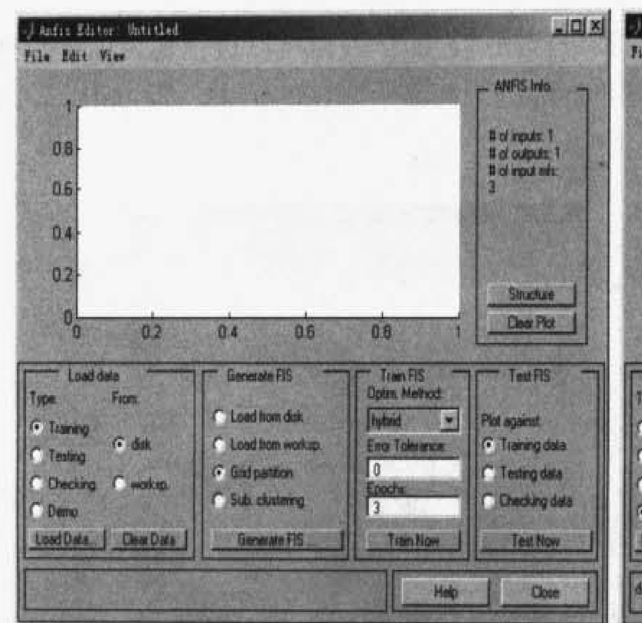


图 6-14　anfis 的图形窗口界面

图 6-15　加载演示数据的 anfis 的图形窗口界面

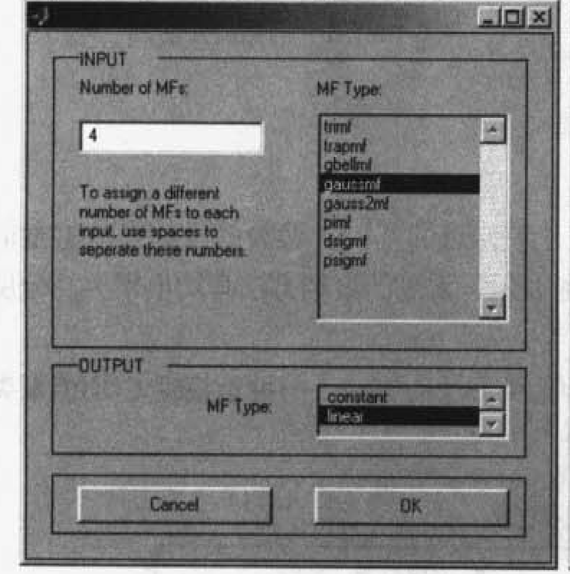


图 6-16　模糊推理系统的对话框

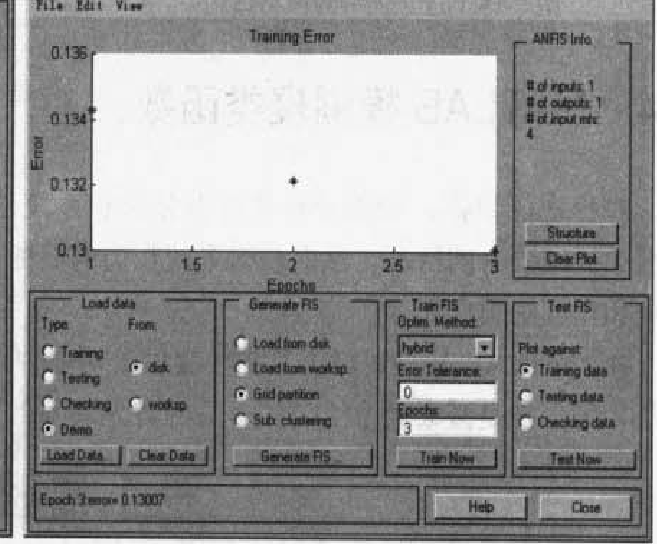


图 6-17　anfis 的训练后的图形界面

3. 训练神经模糊推理系统

在进行神经模糊推理系统的训练前，可以指定优化的方法及有关的优化控制参数。对于前面加载的演示数据，在进行 anfis 的训练后，窗口的上部显示了优化过程中误差的变化情况，如图 6-17 所示。

4. 测试 anfis

在 anfis 的图形界面（见图 6-17）中，利用右上角的【Structure】按钮，可以方便地查看训练得到的 anfis 的网络结构。对应上述演示数据的神经网络结构如图 6-18 所示。

在完成对 anfis 的训练后，可以进一步对其进行测试，测试数据可以被指定为训练数据或另外提供的训练数据。测试完成后，将在图形界面的上部显示测试的结果，即 anfis 输出数据与测试数据的比较，如图 6-19 所示。

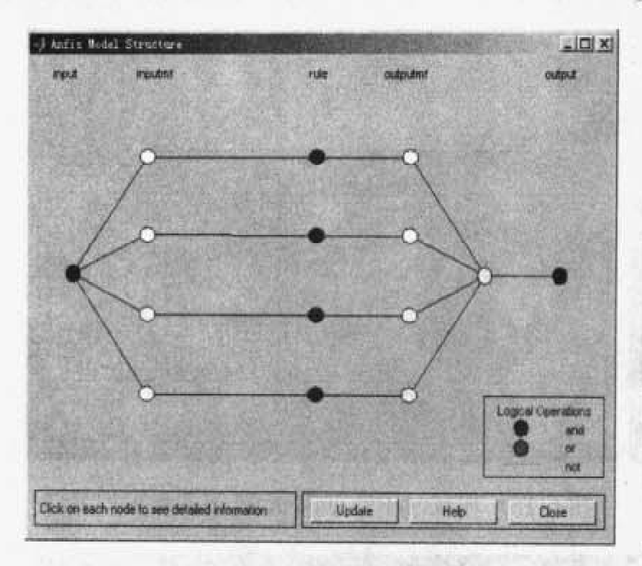


图 6-18 anfis 的神经网络结构

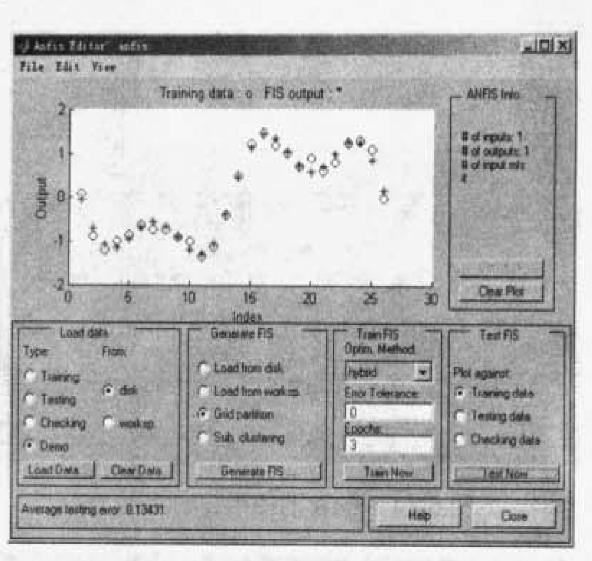


图 6-19 anfis 的测试结果

6.4 MATLAB 模糊聚类函数

对给定的数据，聚类分析是许多分类和系统建模问题的基础。聚类的目的是从大量的数据中抽取固有的特征，从而获得系统行为的简捷表示。常见的聚类方法有均值聚类、分层聚类和模糊聚类方法。

在 MATLAB 模糊逻辑工具箱中提供了对两种聚类方法的支持，一种是模糊 C-均值聚类方法，另一种是减聚类方法。其有关函数见表 6-2。

表 6-2 模糊聚类函数

函 数 名	功 能
fcm()	模糊 C-均值聚类函数
subclust()	减法聚类函数
genfis2()	基于减法聚类的模糊推理系统建模函数

6.4.1 模糊 C-均值聚类函数

在模糊 C-均值聚类方法中，每一个数据点按照一定的模糊隶属度隶属于某一聚类中心。这一聚类技术作为对传统聚类技术的改进，是 Jim Bezdek 于 1981 年提出的。该方法首先随机选取若干聚类中心，所有数据点都被赋予对聚类中心一定的模糊隶属度，然后通过迭代方法不断修正聚类中心，迭代过程以极小化所有数据点到各个聚类中心的距离及隶属度值

的加权和为优化目标。

模糊 C-均值聚类的输出不是一个模糊推理系统，而是聚类中心的列表及每个数据点对各个聚类中心的隶属度值。该输出能够被进一步用来建立模糊推理系统。对应模糊 C-均值聚类方法的函数为 fcm( )，该函数的调用格式为

[center,U,obj_fcn]=fcm(data,cluster_n)

[center,U,obj_fcn]=fcm(data,cluster_n,options)

式中，输入参数 data 为给定的数据集，矩阵 data 的每一行为一个数据点向量；参数 cluster_n 为聚类中心的个数；输入参数向量中的 options 为若干控制参数，这些参数的定义如下：options(1)为分割矩阵的指数（默认值为 2）；options(2)为最大迭代次数（默认值为 100）；options(3)为迭代停止的误差准则（默认值为 1e-5）；options(4)为迭代过程中的信息显示（默认值为 1）；center 为迭代后得到的聚类中心；U 为所有数据点对聚类中心的隶属度函数矩阵；obj_fcn 为目标函数值在迭代过程中的变化值。

聚类过程在达到最大次数或满足停止误差准则时结束。

例 6-3 利用模糊 C-均值聚类方法将一组随机数据分为两类。

解：MATLAB 程序如下：

```
data = rand(100,2);
[center,U,obj_fcn] = fcm(data,2);
plot(data(:,1),data(:,2),'o');
maxU = max(U);
index1 = find(U(1,:) = = maxU);index2 = find(U(2,:) = = maxU);
line(data(index1,1),data(index1,2) ,'linestyle','*','color','r');
line(data(index2,1),data(index2,2) ,'linestyle','o','color','r');
```

其输出结果如图 6-20 所示。

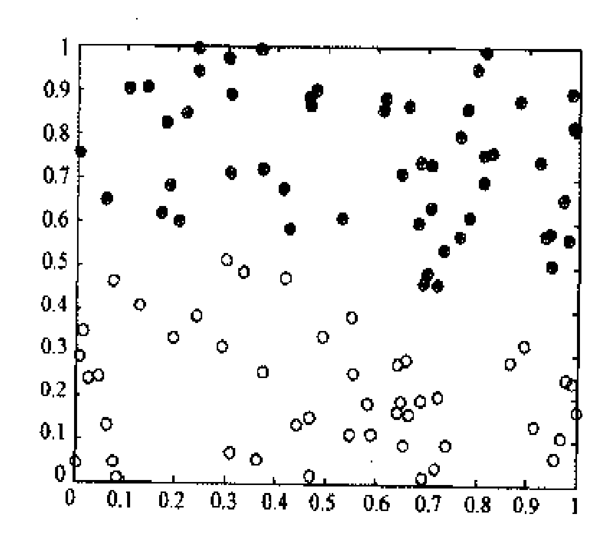


图 6-20 模糊 C-均值分类结果

例 6-4 对给定的数据进行模糊 C-均值聚类。

解：MATLAB 程序如下

```
%加载并显示给定数据
load fcmdata.dat
plot(fcmdata(:,1),fcmdata(:,2),'o')
```

运行以上程序可得如图 6-21 所示结果。fcmdata.dat 为模糊逻辑工具箱自带的一个用于测试的数据文件。

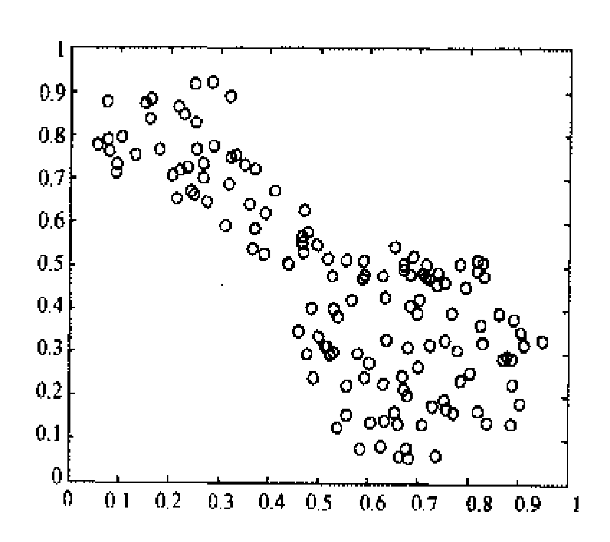


图 6-21 用于模糊 C-均值聚类分析的数据

利用以下程序可得图 6-22 和图 6-23，它们分别显示了聚类过程中目标函数的变化情况和聚类结果。

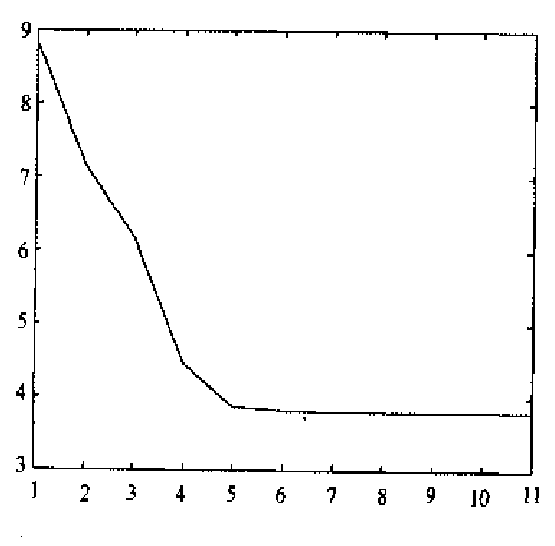


图 6-22 fcm 目标函数变化曲线

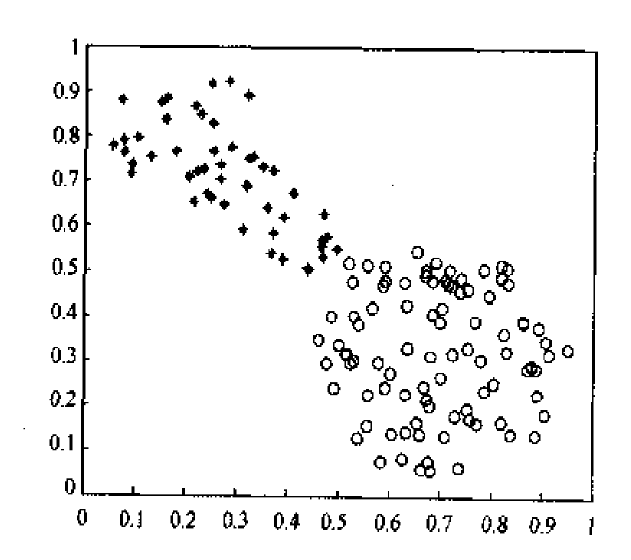


图 6-23 模糊 C-均值聚类结果

```
[center,U,objFcn] = fcm(fcmdata,2);
plot(objFcn)
maxU = max(U);
index1 = find(U(1,:) == maxU);index2 = find(U(2,:) == maxU);
line(fcmdata(index1,1),fcmdata(index1,2),'linestyle','*','color','r');
line(fcmdata(index2,1),fcmdata(index2,2),'linestyle','o','color','r');
```

6.4.2 减法聚类函数

减法聚类方法将每个数据点作为可能的聚类中心，并根据各个数据点周围的数据点密度来计算该点作为聚类中心的可能性。被选为聚类中心的数据点周围具有最高的数据点密度，同时该数据点附近的数据点被排除作为聚类中心的可能性；在选出第一个聚类中心后，从剩余的可能作为聚类中心的数据点中，继续采用类似的方法选择下一个聚类中心。这一过程一直持续到所有剩余的数据点作为聚类中心的可能性低于某一阈值时为止。在 MATLAB 模糊逻辑工具箱中，提供了函数 subclust()，用来完成减法聚类的功能。该函数的调用格式为

[C,S]=subclust(X,radii,XBounds,options)

式中，输入矩阵 X 包含了用于聚类的数据，X 的每一行为一个数据点向量；向量 radii 用于在假定数据点位于一个单位超立方体的条件下，指定数据向量的每一维的聚类中心的影响范围，即 radii 每一维的数值大小均在 0～1 之间，通常的取值范围为 0.2～0.5，若数据点的维数为 2，则 radii=[0.5 0.25]指定了第一维数据的聚类中心的影响范围为数据空间宽度的一半，而第二维数据的聚类中心的影响范围为数据空间宽度的四分之一；XBounds 为一个 2×N 的矩阵，其中 N 为数据的维数。该矩阵用于指定如何将 X 中的数据映射到一个单位超立方体中，其第一行和第二行分别包括了每一维数据被映射到单位超立方体的最小和最大取值。例如，XBounds = [-10 -5;10 5]指定了第一位数据在［-10 10］之间的取值将被映射到［0 1］区间中；而第二维数据相应的区间范围为［-5 5］。若 Xbounds 为空或者没有指定 Xbounds，则默认的映射范围为所有数据点的最小和最大取值构成的区间；参数向量 options 用于指定聚类算法的有关参数，其定义如下：options(1) = squshFactor 为 squshFactor 用于与聚类中心的影响范围 radii 相乘来决定某一聚类中心邻近的哪些数据点被排除作为聚类中心的可能性，默认值为 1.25；options(2) = acceptRatio 为 acceptRatio 用于指定在选出第一个聚类中心后，只有某个数据点作为聚类中心的可能性值高于第一个聚类中心可能性值的一定比例（由 acceptRatio 的大小决定），才能被作为新的聚类中心，默认值为 0.5；options(3) = rejectRatio 为 rejectRatio 用于指定在选出第一个聚类中心后，只有某个数据点作为聚类中心的可能性值低于第一个聚类中心可能性值的一定比例（由 rejectRatio 的大小决定），才能被排除作为聚类中心的可能性，其默认值为 0.15；options(4) = verbose 为若 verbose 为非零

值，则聚类过程的有关信息将显示到窗口中，其默认值为 0；函数的返回值 C 为聚类中心向量，向量 S 包含了数据点每一维聚类中心的影响范围，如以下 MATLAB 命令：

```
>>[C,S] =subclust(X, 0.5)
>>[C,S] = subclust(X, [0.5   0.25   0.3], [ ], [2.0   0.8   0.7] )
```

6.4.3 基于减法聚类的模糊推理系统建模函数

在减法聚类的基础上可以进行模糊推理系统的建模，模糊逻辑工具箱提供的函数 genfis2()能够实现这一功能。该函数的调用格式为

fisMat=genfis2(Xin,Xout,radii,Xbounds,options)

此处给定输入数据 Xin 和输出数据 Xout。该函数首先调用减法聚类函数 subclust( )对输入/输出数据进行减法聚类，以决定输出和输入语言变量的隶属度函数个数和规则个数，并采用最小方差估计得到 Sugeno 型推理规则结论部分的参数。该函数的输出为 Sugeno 型模糊推理系统的矩阵 fisMat；参数 radii、Xbounds 和 options 分别为减法聚类的参数，详细说明参见减法聚类函数 subclust( )，如以下 MATLAB 命令：

```
fisMat = genfis2(Xin, Xout, 0.5)
fisMat = genfis2(Xin, Xout,[ 0.5   0.25   0.3] )
fisMat = genfis2(Xin, Xout, 0.5 ,[-10   -5   0; 10   5   20] )
```

第三篇 预测控制及其 MATLAB 实现

第 7 章 预测控制理论

模型预测控制（Model Predictive Control，MPC）是 20 世纪 80 年代初开始发展起来的一类新型计算机控制算法。该算法直接产生于工业过程控制的实际应用，并在与工业应用的紧密结合中不断完善和成熟。模型预测控制算法采用了多步预测、滚动优化和反馈校正等控制策略，因而具有控制效果好、鲁棒性强、对模型精确性要求不高的优点。由于大量的工业生产过程都具有非线性、不确定性和时变的特点，要建立精确的解析模型十分困难，因此经典控制方法，如 PID 控制及现代控制理论，都难以获得良好的控制效果。而模型预测控制（简称预测控制）具有的优点决定了该方法能够有效地用于复杂工业过程的控制，并且已在石油、化工、冶金、机械等工业部门的过程控制系统中得到了成功的应用。

目前提出的预测控制算法主要有基于非参数模型的模型算法控制（MAC）、动态矩阵控制（DMC）和基于参数模型的广义预测控制（GPC）、广义预测极点配置控制（GPP）等。其中，模型算法控制采用对象的脉冲响应模型，动态矩阵控制采用对象的阶跃响应模型，这两种模型都具有易于获得的优点；广义预测控制和广义预测极点配置控制是预测控制思想与自适应控制的结合，采用 CARIMA 模型（受控自回归积分滑动平均模型），具有参数数目少并能够在线估计的优点，而广义预测极点配置控制进一步采用极点配置技术，提高了预测控制系统的闭环稳定性和鲁棒性。

7.1 动态矩阵控制理论

动态矩阵控制是一种基于计算机控制的技术。它是一种增量算法，并基于系统的阶跃响应，适用于稳定的线性系统，系统的动态特性中具有纯滞后或非最小相位特性都不影响该算法的直接应用。由于它直接以对象的阶跃响应离散系数为模型，避免了通常的传递函数或状态空间方程模型参数的辨识，又由于采用多步预估技术，能有效地解决时延过程问题，并按使预估输出与给定值偏差最小的二次性能指标实施控制，因此是一种最优控制技术。动态矩阵控制算法的控制结构主要由预测模型、滚动优化、误差校正和闭环控制形式构成。

7.1.1 预测模型

从被控对象的阶跃响应出发，对象动态特性用一系列动态系数 $a_1,a_2,\cdots,a_p$，即单位阶

跃响应在采样时刻的值来描述，p 称为模型时域长度，a_p 是足够接近稳态值的系数，如图 7-1 所示。

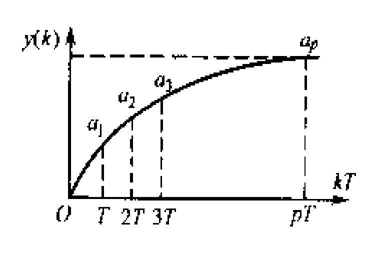


图 7-1 单位阶跃响应曲线

根据线性系统的比例和叠加性质（系数不变原理），若在某个 $k-i(k\geqslant i)$ 时刻输入 $u(k-i)$，则 $\Delta u(k-i)$ 对输出 $y(k)$ 的贡献为

$$y(k)=\begin{cases}a_i\Delta u(k-i) & 1\leqslant i<p\\ a_p\Delta u(k-i) & i\geqslant p\end{cases} \tag{7-1}$$

若在所有 $k-i(i=1,2,\cdots,k)$ 时刻同时有输入，则跟据叠加原理有

$$y(k)=\sum_{i=1}^{p-1}a_i\Delta u(k-i)+a_p\Delta u(k-p) \tag{7-2}$$

利用上式容易得到 $y(k+j)$ 的 n 步预估($n<p$)为

$$\hat{y}(k+j)=\sum_{i=1}^{p-1}a_i\Delta u(k+j-i)+a_p\Delta u(k+j-p)\quad (j=1,2,\cdots,n) \tag{7-3}$$

由于只有过去的控制输入是已知的，因此在利用动态模型作预估时有必要把过去的输入对未来的输出贡献分离出来，上式可写为

$$\hat{y}(k+j)=\sum_{i=1}^{j}a_i\Delta u(k+j-i)+\sum_{i=j+1}^{p-1}a_i\Delta u(k+j-i)+a_p\Delta u(k+j-p)$$

$$(j=1,2,\cdots,n) \tag{7-4}$$

上式右端的后二项即为过去输入对输出 n 步预估，记为

$$y_0(k+j)=\sum_{i=j+1}^{p-1}a_i\Delta u(k+j-i)+a_p\Delta u(k+j-p)\quad (j=1,2,\cdots,n) \tag{7-5}$$

将式（7-4）写成矩阵形式

$$\begin{bmatrix}\hat{y}(k+1)\\ \hat{y}(k+2)\\ \cdots\\ \hat{y}(k+n)\end{bmatrix}=\begin{bmatrix}a_1 & & & \mathbf{0}\\ a_2 & a_1 & & \\ \vdots & \vdots & \ddots & \\ a_n & a_{n-1} & \cdots & a_1\end{bmatrix}\begin{bmatrix}\Delta u(k)\\ \Delta u(k+1)\\ \vdots\\ \Delta u(k+n-1)\end{bmatrix}+\begin{bmatrix}y_0(k+1)\\ y_0(k+2)\\ \vdots\\ y_0(k+n)\end{bmatrix} \tag{7-6}$$

为增加系统的动态稳定性和控制输入的可实现性，并减少计算量，可将 Δu 向量减少为 m 维($m<n$)，则式（7-6）变为

$$\begin{bmatrix}\hat{y}(k+1)\\ \hat{y}(k+2)\\ \vdots\\ \hat{y}(k+n)\end{bmatrix}=\begin{bmatrix}a_1 & & & \mathbf{0}\\ a_2 & a_1 & & \\ \vdots & \vdots & \ddots & \\ a_n & a_{n-1} & \cdots & a_{n-m+1}\end{bmatrix}\begin{bmatrix}\Delta u(k)\\ \Delta u(k+1)\\ \vdots\\ \Delta u(k+m-1)\end{bmatrix}+\begin{bmatrix}y_0(k+1)\\ y_0(k+2)\\ \vdots\\ y_0(k+n)\end{bmatrix} \tag{7-7}$$

令 $\hat{Y}=[\hat{y}(k+1),\hat{y}(k+2),\cdots,\hat{y}(k+n)]^{\mathrm{T}}$

$$\Delta \boldsymbol{U} = [\Delta u(k), \Delta u(k+1), \cdots, \Delta u(k+m-1)]^{\mathrm{T}}$$

$$\boldsymbol{Y}_0 = [y_0(k+1), y_0(k+2), \cdots, y_0(k+n)]^{\mathrm{T}}$$

$$\boldsymbol{A} = \begin{bmatrix} a_1 & & & \boldsymbol{0} \\ a_2 & a_1 & & \\ \vdots & \vdots & \ddots & \\ a_n & a_{n-1} & \cdots & a_{n-m+1} \end{bmatrix}$$

则式（7-7）可写为

$$\hat{\boldsymbol{Y}} = \boldsymbol{A}\Delta \boldsymbol{U} + \boldsymbol{Y}_0 \tag{7-8}$$

式中，矩阵 $\boldsymbol{A}$ 为 $n \times m$ 维的常数矩阵，由于它完全由系统的阶跃响应参数所决定，反映了对象的动态特性，故称之为动态矩阵；n 和 m 分别称之为最大预测长度和控制长度。

7.1.2　滚动优化

系统的模型预测是根据动态响应系数和控制增量来决定的，该算法的控制增量是通过使最优化准则

$$J = \sum_{j=1}^{n}[y(k+j) - w(k+j)]^2 + \sum_{j=1}^{m}\lambda(j)[\Delta u(k+j-1)]^2 \tag{7-9}$$

的值为最小来确定的，以使系统在未来 $n(p \leqslant n \leqslant m)$ 个时刻的输出值尽可能接近期望值。为简单起见，取控制加权系数 $\lambda(j) = \lambda$（常数）。

若令

$$\boldsymbol{W} = [w(k+1), w(k+2), \cdots, w(k+n)]^{\mathrm{T}}$$

则式（7-9）可表示为

$$J = (\boldsymbol{Y} - \boldsymbol{W})^{\mathrm{T}}(\boldsymbol{Y} - \boldsymbol{W}) + \lambda \Delta \boldsymbol{U}^{\mathrm{T}} \Delta \boldsymbol{U} \tag{7-10}$$

式中，$w(k+j)$ 称为期望输出序列值，在预测控制这类算法中，要求闭环响应沿着一条指定的、平滑的曲线到达新的稳定值，以提高系统的鲁棒性。

一般取

$$w(k+j) = \alpha^j y(k) + (1-\alpha^j) y_r \qquad (j = 1, 2, \cdots, n)$$

式中，α 为柔化系数，$0 < \alpha < 1$；$y(k)$ 为系统实测输出值；y_r 为系统的给定值。

用 $\boldsymbol{Y}$ 的最优预测值 $\hat{\boldsymbol{Y}}$ 代替 $\boldsymbol{Y}$，即将式（7-8）代入式（7-10）中

并令　$\dfrac{\partial J}{\partial \Delta \boldsymbol{U}} = 0$

得

$$\Delta \boldsymbol{U} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \lambda \boldsymbol{I})^{-1}\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{W} - \boldsymbol{Y}_0) \tag{7-11}$$

式（7-11）与实际检测值无关，它是 DMC 算法的开环控制形式。由于受模型误差、弱非线性特性等影响，开环控制式（7-11）不能紧密跟踪期望值，若等到经过 m 个时刻后，再重复式（7-11），则必然造成较大的偏差，更不能抑制系统受到的扰动，故必须采用闭环

控制算式，即仅将计算出来的 m 个控制增量的第一个值付诸实施，即 k 时刻的控制增量为

$$\Delta u(k)=\boldsymbol{c}^{\mathrm{T}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}+\lambda\boldsymbol{I})^{-1}\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{W}-\boldsymbol{Y}_0)=\boldsymbol{d}^{\mathrm{T}}(\boldsymbol{W}-\boldsymbol{Y}_0) \tag{7-12}$$

式中，$\boldsymbol{c}^{\mathrm{T}}=[1\quad 0\quad \cdots\quad 0]$；$\boldsymbol{d}^{\mathrm{T}}=\boldsymbol{c}^{\mathrm{T}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}+\lambda\boldsymbol{I})^{-1}\boldsymbol{A}^{\mathrm{T}}$

若 $\boldsymbol{A}$、λ 都已确定，$\boldsymbol{d}$ 可事先离线解出，则在线计算$\Delta u(k)$只需完成两个矢量的点积即可。

可见，预测控制的控制策略是在实施了$\Delta u(k)$之后，采集 k+1 时刻的输出数据，进行新的预测、校正、优化，从而避免在等待 m 拍控制输入完毕期间，因受干扰等影响造成的失控。因此优化过程不是一次离线进行，而是反复在线进行的，其优化目标也是随时间推移的，即在每一时刻都提出一个立足于该时刻的局部优化目标，而不是采用不变的全局优化目标。

7.1.3 误差校正

由于每次实施控制，只采用了第一个控制增量$\Delta u(k)$，故对未来时刻的输出可用下式预测

$$\hat{\boldsymbol{Y}}_p=\boldsymbol{a}\Delta u(k)+\boldsymbol{Y}_{p0} \tag{7-13}$$

式中，$\hat{\boldsymbol{Y}}_p=[\hat{y}(k+1),\hat{y}(k+2),\cdots,\hat{y}(k+p)]^{\mathrm{T}}$ 表示在 t=kT 时刻预测的有 $\Delta u(k)$ 作用时的未来 p 个时刻的系统输出；$\boldsymbol{Y}_{p0}=[y_0(k+1),y_0(k+2),\cdots,y_0(k+p)]^{\mathrm{T}}$ 表示在 t=kT 时刻预测的无 $\Delta u(k)$ 作用时的未来 p 个时刻的系统输出；$\boldsymbol{a}=[a_1,a_2,\cdots,a_p]^{\mathrm{T}}$ 为单位阶跃响应在采样时刻的值。

由于对象及环境的不确定性，在 k 时刻实施控制作用后，在 k+1 时刻的实际输出 $y(k+1)$ 与预测的输出

$$\hat{y}(k+1)=y_0(k+1)+a_1\Delta u(k)$$

未见得相等，这就需要构成预测误差

$$e(k+1)=y(k+1)-\hat{y}(k+1)$$

并用此误差加权后修正对未来其他时刻的预测

即

$$\tilde{\boldsymbol{Y}}_p=\hat{\boldsymbol{Y}}_p+\boldsymbol{h}e(k+1) \tag{7-14}$$

式中，$\tilde{\boldsymbol{Y}}_p=[\tilde{y}(k+1),\tilde{y}(k+2),\cdots,\tilde{y}(k+p)]^{\mathrm{T}}$ 为 t=(k+1)T 时刻经误差校正后所预测的 t=(k+1)T 时刻的系统输出；$\boldsymbol{h}=[h_1,h_2,\cdots,h_p]^{\mathrm{T}}$ 为误差校正矢量，$h_1=1$。

经校正后的$\tilde{\boldsymbol{Y}}_p$ 作为下一时刻的预测初值，由于利用在 t=(k+1)T 时刻的预测初值预测 t=(k+2)T,…,(k+p+1)T 时刻的输出值，故令

$$y_0(k+i)=\tilde{y}(k+i+1)\qquad(i=1,2,\cdots,p-1) \tag{7-15}$$

由式（7-14）和式（7-15）得下一时刻的预测初值为

$$\begin{cases} y_0(k+i)=\hat{y}(k+i+1)+h_{i+1}e(k+1) \\ \qquad\qquad (i=1,2,\cdots,p-1) \\ y_0(k+p)=\hat{y}(k+p)+h_p e(k+1) \end{cases} \tag{7-16}$$

这一修正的引入，也使系统成为一个闭环负反馈系统，对提高系统的性能起了很大作用。

由此可见，动态矩阵控制是由预测模型、控制器和校正器三部分组成的，模型的功能在于预测未来的输出值，控制器则决定了系统输出的动态特性，而校正器则只有当预测误差存在时才起作用。

7.2 广义预测控制理论

十多年来产生了许多自校正器，都成功地用于实际过程，但对变时延、变阶次和变参数过程，控制效果不好。因此研制具有鲁棒性的自校正器成为人们关注的问题。

Richalet 等人提出了大范围预测概念，在此基础上，Clarke 等人提出了广义预测自校正器，该算法以 CARIMA 模型为基础，采用了长时段的优化性能指标，结合辨识和自校正机制，具有较强的鲁棒性和模型要求低等特点，并有广泛的适用范围。这个算法可克服广义最小方差（需要试凑控制量的加权系数）、极点配置（对阶的不确定性十分敏感）等自适应算法中存在的缺点。近年来，它在国内外控制理论界已引起了广泛的重视，GPC 法可看成是迄今所知的自校正控制方法中最为接近具有鲁棒性的一种。

广义预测控制是一种新的远程预测控制方法，概括起来具有以下特点：

① 基于 CARIMA 模型；

② 目标函数中对控制增量加权的考虑；

③ 利用输出的远程预报；

④ 控制时域长度概念的引入；

⑤ 丢番图方程的递推求解。

7.2.1 预测模型

在预测控制理论中，需要有一个描述系统动态行为的基础模型，称为预测模型。它应具有预测的功能，即能够根据系统的历史数据和未来的输入，预测系统未来输出值。GPC 采用 CARIMA 模型作为预测模型，模型 CARIMA 是“Controlled Auto-Regressive Integrated Moving-Average”的缩写，可以译为“受控自回归积分滑动平均模型”。这个模型可以写成

$$A(z^{-1})y(k)=B(z^{-1})u(k-1)+C(z^{-1})\xi(k)/\Delta \tag{7-17}$$

式中，$A(z^{-1})$,$B(z^{-1})$和 $C(z^{-1})$分别是 n、m 和 n 阶的 z^{-1} 的多项式，$\Delta=1-z^{-1}$；$y(k)$、$u(k)$和 $\xi(k)$分别表示输出、输入和均值为零的白噪声序列，若系统时滞大于零，则 $B(z^{-1})$多项式开

头的一项或几项系数等于零。Clarke 等人在推导广义预测控制时，为了简单起见，令 $C(z^{-1})=1$。

CARIMA 模型具有下列优点：

① 模型中的积分作用可消除余差；

② 可适用于一类非平稳随机噪声过程；

③ 可以描述能用 ARMAX 模型描述的过程；

④ 在大多数情况下，CARIMA 模型比 ARMAX 模型辨识效果更好；

⑤ 用 CARIMA 模型导出的控制器对未建模动态具有较好的鲁棒性。

7.2.2 滚动优化

1．目标函数

为增强系统的鲁棒性（Robustness），在目标函数中考虑了现在时刻的控制 $u(k)$ 对系统未来时刻的影响，采用下列目标函数

$$J=\sum_{j=1}^{n}[y(k+j)-w(k+j)]^2+\sum_{j=1}^{m}\lambda(j)[\Delta u(k+j-1)]^2 \tag{7-18}$$

式中，n 称为最大预测长度，一般应大于 $B(z^{-1})$ 的阶数；m 表示控制长度($m\leqslant n$)；$\lambda(j)$ 是大于零的控制加权系数。为简单起见，取 $\lambda(j)=\lambda$（常数）。

为了进行柔化控制，控制的目的不是使输出直接跟踪设定值，而是跟踪参考轨线，参考轨线由下式产生

$$w(k+j)=\alpha^j y(k)+(1-\alpha^j)y_r \qquad (j=1,2,\cdots,n) \tag{7-19}$$

式中，y_r、$y(k)$ 和 $w(k+j)$ 分别为设定值、输出和参考轨线；α 为柔化系数，$0<\alpha<1$。

目标函数中后一项的加入，主要用于压制过于剧烈的控制增量，以防止系统超出限制范围或发生剧烈振荡。

广义预测控制问题，可以归结为求 $\Delta u(k)$, $\Delta u(k+1)$, $\cdots$, $\Delta u(k+m-1)$ 使得目标函数式（7-18）达到最小值，这是一个优化问题。

2．输出预测

根据预测理论，为了预测超前 j 步输出，引入丢番图 Dioaphantine 方程

$$1=E_j(z^{-1})A(z^{-1})\Delta+z^{-j}F_j(z^{-1}) \tag{7-20}$$

式中，$E_j(z^{-1})=e_{j0}+e_{j1}z^{-1}+\cdots+e_{j,j-1}z^{-j+1}$;　　$e_{j0}=1$;

$F_j(z^{-1})=f_{j0}+f_{j1}z^{-1}+\cdots+f_{jn}z^{-n}$。

将式（7-17）两边同时乘以 $E_j(z^{-1})\Delta$ 后与式（7-20）可得时刻 k 后 j 步的预测方程为

$$y(k+j)=E_j(z^{-1})B(z^{-1})\Delta u(k+j-1)+F_j(z^{-1})y(k)+E_j(z^{-1})\xi(k+j) \tag{7-21}$$

令　$G_j(z^{-1}) = E_j(z^{-1})B(z^{-1})$

$$= g_{j0} + g_{j1}z^{-1} + \cdots + g_{jj-1}z^{-j+1} + g_{jj}z^{-j} + \cdots$$

注意到　$G_j(z^{-1}) = E_j(z^{-1})B(z^{-1}) = \dfrac{B(z^{-1})}{A(z^{-1})\Delta}[1 - z^{-j}F_j(z^{-1})]$

可知$G_j(z^{-1})$的前j项正是系统单位阶跃响应$g_0, g_1, \cdots$的前j项。

故有　$G_j(z^{-1}) = g_0 + g_1 z^{-1} + \cdots + g_{j-1}z^{-j+1} + g_{jj}z^{-j} + \cdots$

为书写方便，将式（7-21）简记为

$$y(k+j) = G_j(z^{-1})\Delta u(k+j-1) + F_j(z^{-1})y(k) + E_j(z^{-1})\xi(k+j) \tag{7-22}$$

对未来输出值的预报，可忽略未来噪声的影响，得到

$$\hat{y}(k+j) = G_j(z^{-1})\Delta u(k+j-1) + F_j(z^{-1})y(k) \quad (j = 1,2,\cdots,n) \tag{7-23}$$

上式中包括k时刻的已知量和未知量两部分，用$f(k+j)$表示已知量，即

$$\begin{cases} f(k+1) = (G_1 - g_0)\Delta u(k) + F_1 y(k) \\ f(k+2) = z(G_2 - z^{-1}g_1 - g_0)\Delta u(k) + F_2 y(k) \\ \quad \cdots \\ f(k+n) = z^{n-1}(G_n - z^{-n+1}g_{n-1} - \cdots - z^{-1}g_1 + g_0)\Delta u(k) + F_n y(k) \end{cases}$$

写成矩阵形式

$$f = H\Delta u(k) + Fy(k) \tag{7-24}$$

式中，

$$H = \begin{bmatrix} G_1 - g_0 \\ z(G_2 - z^{-1}g_1 - g_0) \\ \vdots \\ z^{n-1}(G_n - z^{-n+1}g_{n-1} - \cdots - z^{-1}g_1 + g_0) \end{bmatrix} = \begin{bmatrix} g_{11}z^{-1} + g_{12}z^{-2} + \cdots \\ g_{22}z^{-1} + g_{23}z^{-2} + \cdots \\ \vdots \\ g_{nn}z^{-1} + g_{n(n+1)}z^{-2} + \cdots \end{bmatrix}$$

$$F = [F_1, F_2, \cdots, F_n]^{\mathrm{T}}$$

根据式（7-23），可得最优输出预测值为

$$\hat{Y} = G\Delta U + f \tag{7-25}$$

式中，$\hat{Y} = [\hat{y}(k+1), \hat{y}(k+2), \cdots, \hat{y}(k+n)]^{\mathrm{T}}$

$$\Delta U = [\Delta u(k), \Delta u(k+1), \cdots, \Delta u(k+n-1)]^{\mathrm{T}}$$

$$f = [f(k+1), f(k+2), \cdots, f(k+n)]^{\mathrm{T}}$$

$$G = \begin{bmatrix} g_0 & & & \mathbf{0} \\ g_1 & g_0 & & \\ \vdots & \vdots & \ddots & \\ g_{n-1} & g_{n-2} & \cdots & g_0 \end{bmatrix}$$

3. 最优控制律

若令 $W=[w(k+1),w(k+2),\cdots,w(k+n)]^{\mathrm T}$，

则式（7-18）可表示为

$$J=(Y-W)^{\mathrm T}(Y-W)+\lambda\Delta U^{\mathrm T}\Delta U \tag{7-26}$$

用 Y 的最优预测值 $\hat{Y}$ 代替 Y，即将式（7-25）代入式（7-26），

并令

$$\frac{\partial J}{\partial \Delta U}=0$$

可得

$$\Delta U=(G^{\mathrm T}G+\lambda I)^{-1}G^{\mathrm T}(W-f) \tag{7-27}$$

实际控制时，每次仅将第一个分量加入系统，即

$$u(k)=u(k-1)+g^{\mathrm T}(W-f) \tag{7-28}$$

式中，$g^{\mathrm T}$ 为 $(G^{\mathrm T}G+\lambda I)^{-1}G^{\mathrm T}$ 的第一行。

为了计算简单，通常选 $m<n$，相当于令 $j>m$ 时，$\Delta u(k+j-1)=0$。这时，Δu 变成了 $m\times1$ 矩阵，$(G^{\mathrm T}G+\lambda I)$ 则变成 $m\times m$ 方阵，降低了维数，减小了计算量。对于阶数较低的简单系统，取 $m=1$，则整个过程将不包括任何矩阵运算。

与通常的最优控制不同，广义预测控制采用滚动优化，优化目标是随时间推移的，即在每一时刻都提出一个立足于该时刻的局部优化目标，而不是采用不变的全局优化目标。因此，优化过程不是一次离线进行，而是反复在线进行的，这种滚动优化目标的局部性，使其在理想条件下，只能得到全局的次优解。然而，当模型失配或有时变、非线性及干扰影响时，它却能顾及这种不确定性，及时进行弥补，减小偏差，保持实际上的最优。

7.2.3 反馈校正

在广义预测控制算法推导过程中，虽然没有明显给出反馈或闭环的表示，但它在进行滚动优化时，强调了优化的基点与实际系统一致。这意味着在控制的每一步，都要检测实际输出并与预测值比较，并以此修正预测的不确定性。当实际系统存在非线性、时变、模型失配、干扰等因素时，这种反馈校正就能及时修正预测值，使优化建立在较准确的预测基础上。因此可降低对基础模型的要求，提高控制的鲁棒性，在实际工业应用中，这点具有十分现实的意义。

7.3 预测控制理论分析

7.3.1 广义预测控制的性能分析

1. 引言

近年来，广义预测控制无论是在工业应用，还是在控制理论界都得到了广泛的重视，

然而由于优化的启发性和算法的复杂性，对于这一算法的理论研究十分困难。本节通过系统的闭环方块图求出闭环传递函数，并在内模控制结构的基础上，分析系统的闭环动态特性、稳定性和鲁棒性。

2. 广义预测控制系统闭环传递函数

1）闭环方块图

为推求系统的闭环方块图，须将GPC算法各部分稍加变换并重写如下。

（1）预测模型式（7-17）

$$y(k)=\frac{z^{-1}\boldsymbol{B}}{\boldsymbol{A}}u(k)+\frac{\boldsymbol{C}}{\boldsymbol{A}\Delta}\xi(k) \tag{7-29}$$

（2）预测向量 $\boldsymbol{f}$ 式（7-24）

$$\boldsymbol{f}=\boldsymbol{H}\Delta u(k)+\boldsymbol{F}y(k) \tag{7-30}$$

（3）参考轨迹 $\boldsymbol{W}$ 式（7-19）

$$\boldsymbol{W}=\boldsymbol{Q}y(k)+\boldsymbol{M}y_r \tag{7-31}$$

式中，$\boldsymbol{W}=[w(k+1),w(k+2),\cdots,w(k+n)]^{\mathrm{T}}$；

$\boldsymbol{Q}=[\alpha,\alpha^2,\cdots,\alpha^n]^{\mathrm{T}}$；

$\boldsymbol{M}=[1-\alpha,1-\alpha^2,\cdots,1-\alpha^n]^{\mathrm{T}}$

（4）控制增量 $\Delta u(k)$ 式（7-28）

$$\Delta u(k)=\boldsymbol{g}^{\mathrm{T}}(\boldsymbol{W}-\boldsymbol{f}) \tag{7-32}$$

由以上4式得闭环方块图，如图7-2所示。

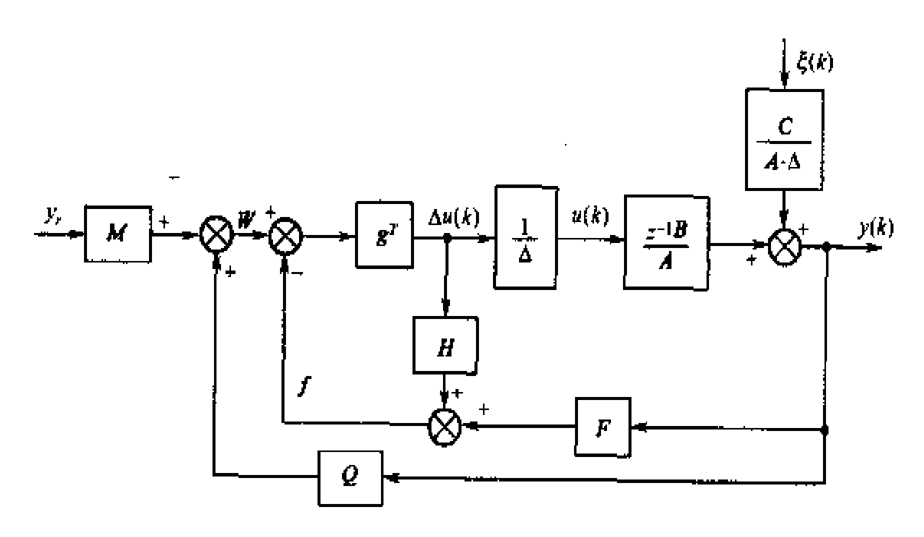


图7-2　GPC控制系统闭环方块图

2）控制结构图

根据如图7-2所示的闭环方块图，可画出GPC的控制结构图，如图7-3所示。

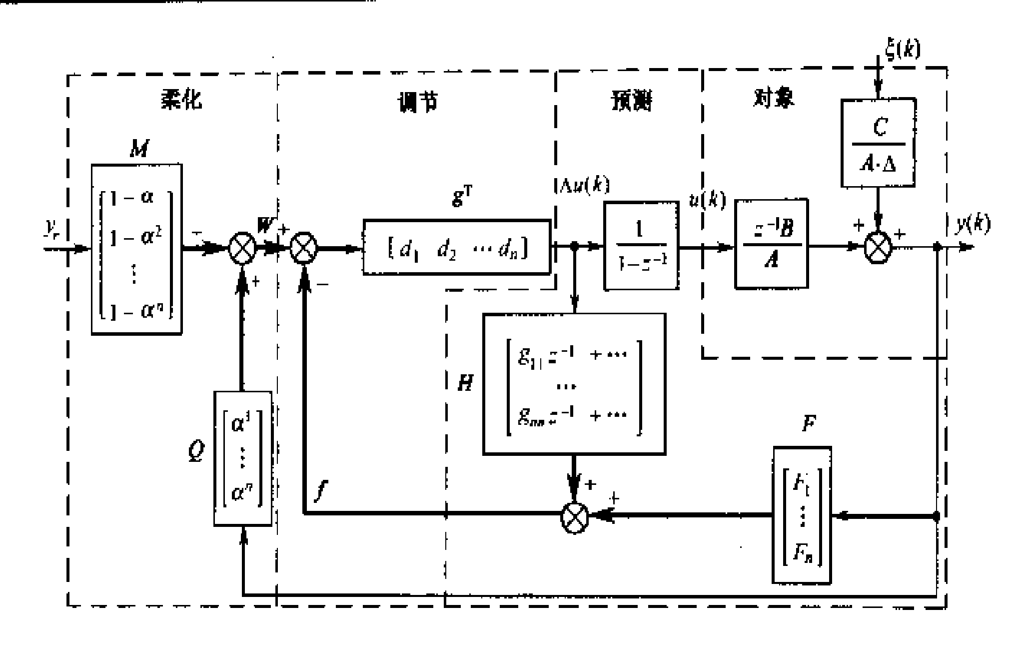


图 7-3 GPC 控制结构图

图中粗线表示矢量信号流，细线表示标量信号流。由此可见，GPC 是由柔化、调节和预测三部分构成的。在每一时刻，给定值序列经过柔化作用后得到的期望输出向量，与预测输出相比较构成偏差向量，偏差向量与动态向量 $\boldsymbol{g}^{\mathrm{T}}$ 点积得到该时刻的控制增量 $\Delta u(k)$，控制增量一方面通过数字积分运算求出控制量 $u(k)$ 作用于对象，另一方面又与系统输出一起去预测新的系统输出值 $\boldsymbol{f}$。

3）闭环传递函数

由图 7-2 可求出 GPC 系统的闭环传递函数为

$$\frac{y(k)}{y_r}=\frac{z^{-1}B\boldsymbol{g}^{\mathrm{T}}\boldsymbol{M}}{(1+\boldsymbol{g}^{\mathrm{T}}\boldsymbol{H})A\Delta+z^{-1}B\boldsymbol{g}^{\mathrm{T}}(\boldsymbol{F}-\boldsymbol{Q})} \tag{7-33}$$

闭环系统的输出响应为

$$y(k)=\frac{z^{-1}B\boldsymbol{g}^{\mathrm{T}}\boldsymbol{M}}{(1+\boldsymbol{g}^{\mathrm{T}}\boldsymbol{H})A\Delta+z^{-1}B\boldsymbol{g}^{\mathrm{T}}(\boldsymbol{F}-\boldsymbol{Q})}y_r+\frac{(1+\boldsymbol{g}^{\mathrm{T}}\boldsymbol{H})C}{(1+\boldsymbol{g}^{\mathrm{T}}\boldsymbol{H})A\Delta+z^{-1}B\boldsymbol{g}^{\mathrm{T}}(\boldsymbol{F}-\boldsymbol{Q})}\xi(k) \tag{7-34}$$

由以上可得如下结论：

（1）GPC 不是用控制器的极点去对消对象的零点，因而不存在因对消不精确所带来的不稳定性问题，故 GPC 可用于非最小相位系统。

（2）闭环传递函数中不包括 $C(z^{-1})$，只要 $C(z^{-1})$ 是稳定的多项式，至于 $C(z^{-1})$ 的精度，它只影响跟踪性能而不影响闭环稳定性和鲁棒性。

（3）当 $k\to\infty(z\to1)$ 时，$\Delta\to0$，而 $E\{\xi(k)\}\to0$，由式（7-20）知 $\boldsymbol{F}_j\to1$，将

$F \to M + Q$ 代入式（7-34）可得 $y(k) \to y_r$，即系统在稳态时无差跟踪设定值，即使在有阶跃扰动情况下也是无差跟踪，这是由 CARIMA 模型内部积分作用所决定的。

（4）GPC 系统闭环传递函数形式较为复杂（涉及到向量 F 和 H），不能明显看出 GPC 的设计参数对系统性能的关系，不便进行系统分析。为此，将 GPC 算法结构变为内模结构形式，以便借助于内模原理研究系统的鲁棒性、稳定性及其他特性。

3. 广义预测控制的内模控制描述

1）GPC 内模结构的推导过程

由图 7-2 依次变换可得图 7-4（a）、（b）和（c）。

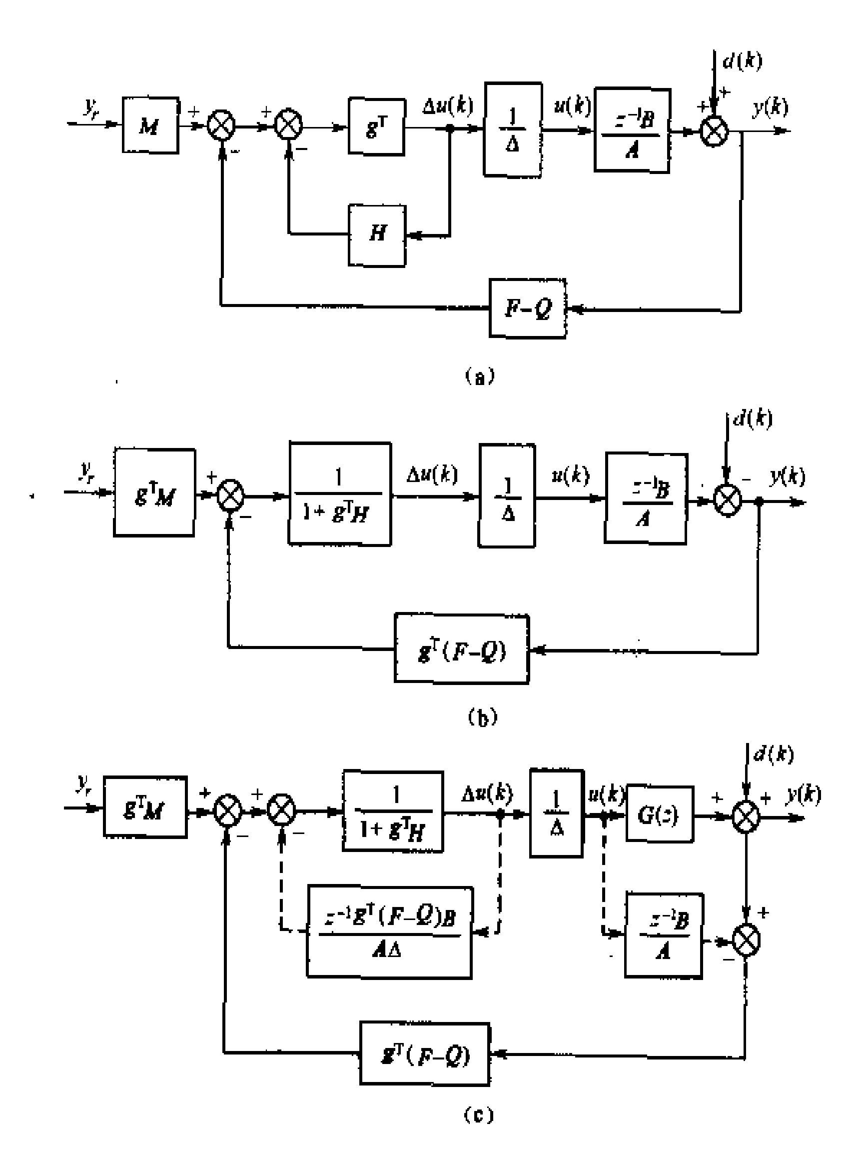


图 7-4　GPC 系统内模结构的推导过程

最后，由图 7-4（c）可得 GPC 的内模结构图，如图 7-5 所示。

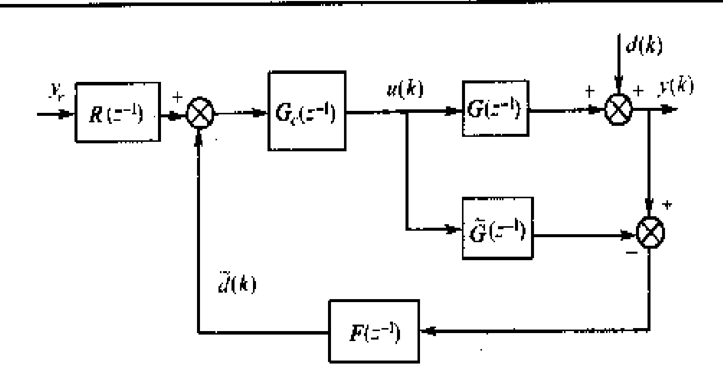


图 7-5 GPC 系统内模结构图

图中，$G(z^{-1})$ 为被控过程的实际模型；$\tilde{G}(z^{-1})$ 为被控过程的预测模型；$G_c(z^{-1})$ 为控制器；$F(z^{-1})$ 为滤波器；$d(k)$ 为总的扰动量；$\tilde{d}(k)$ 为模型误差信号。

其中，

$$\tilde{G}(z^{-1})=\frac{z^{-1}B}{A},G_c(z^{-1})=\frac{A}{(1+g^{\mathrm{T}}H)A\Delta+z^{-1}g^{\mathrm{T}}(F-Q)B}$$

$$F(z^{-1})=g^{\mathrm{T}}(F-Q)=\sum_{j=1}^{n}d_j(F_j-\alpha^j) \tag{7-35}$$

$$R(z^{-1})=g^{\mathrm{T}}M=\sum_{j=1}^{n}d_j(1-\alpha^j),g^{\mathrm{T}}=[d_1,d_2,\cdots,d_n]$$

2）GPC 的闭环稳定性

由系统的内模结构图 7-5 可得系统的控制量和输出量分别为

$$u(k)=\frac{G_c(z^{-1})}{1+G_c(z^{-1})F(z^{-1})[G(z^{-1})-\tilde{G}(z^{-1})]}[R(z^{-1})y_r-F(z^{-1})d(k)] \tag{7-36}$$

$$y(k)=\frac{G(z^{-1})G_c(z^{-1})}{1+G_c(z^{-1})F(z^{-1})[G(z^{-1})-\tilde{G}(z^{-1})]}[R(z^{-1})y_r-F(z^{-1})d(k)]+d(k) \tag{7-37}$$

系统稳定的充要条件是闭环特征方程的零点位于单位圆内。

由式（7-36）和式（7-37）得特征方程

$$\frac{1}{G_c(z^{-1})}+F(z^{-1})[G(z^{-1})-\tilde{G}(z^{-1})]=0 \tag{7-38}$$

$$\frac{1}{G(z^{-1})G_c(z^{-1})}+\frac{F(z^{-1})}{G(z^{-1})}[G(z^{-1})-\tilde{G}(z^{-1})]=0 \tag{7-39}$$

当模型匹配 $G(z)=\tilde{G}(z^{-1})$ 时，由式（7-38）和式（7-39）得

$$\frac{1}{G_c(z^{-1})}=0 \tag{7-40}$$

$$\frac{1}{G(z^{-1})G_c(z^{-1})}=0 \tag{7-41}$$

由此可得结论如下：

（1）若对象稳定，控制器稳定，则闭环系统稳定；

（2）尽管控制器具有非线性特性，但只要保证输入/输出的稳定性，对稳定的受控对象，一定能得到稳定的闭环响应，也就是说，当系统输入 $u(k)$ 受到约束时，不会影响整个系统的稳定性，这一优点深受广大工程人员的欢迎；

（3）若对象开环不稳定，除非有准确的极点对消，否则不能得到稳定解。

以上结论（3）说明，如果对消不完全，不稳定因素总会被激励出来。GPC 解决这个问题的措施在于控制时域长度概念的引入，即经过一段时间 m 后，令控制增量为零，这相当在目标函数中对控制增量作无穷大加权，正是这个无穷大加权，才使不稳定对消因子的作用受到抑制。

3）GPC 系统的鲁棒性

这里所谈的鲁棒性是指当模型失配 $G(z^{-1})\neq\tilde{G}(z^{-1})$ 时，系统仍能维持一定的稳定性的程度。

基于模型 CARIMA 的广义预测控制，由于采用了下列措施，因此增强了系统的鲁棒性（Robustness）。

（1）参考轨迹的引入

为了使控制过程平稳，不要求系统输出 $y(k)$ 直接跟踪设定值 y_r，而使对象输出 $y(k)$ 沿着参考轨迹到达给定值 y_r。

由式（7-19）容易看出，α 小，$w(k)$ 很快趋向 y_r，这时控制系统跟踪的快速性好，但鲁棒性差；增大 α，$w(k)$ 跟踪 y_r 的过程较长，这时系统跟踪的快速性变差而鲁棒性提高。在实际应用中，α 值的选择是控制系统的快速性要求与鲁棒性要求的折中。

（2）在目标函数中考虑了现在时刻的控制 $u(k)$ 对系统将来输出的影响

广义预测控制问题，可以归结为求 $\Delta u(k)$，$\Delta u(k+1)$，…，$\Delta u(k+m-1)$，使得目标函数式（7-18）达到最小值，这是一个最优化问题。下面借助内模原理对这些措施作一分析。

① 由式（7-38）的特征方程

$$\frac{1}{G_c(z^{-1})}+F(z^{-1})[G(z^{-1})-\tilde{G}(z^{-1})]=0$$

可见，当 $G(z^{-1})\neq\tilde{G}(z^{-1})$ 时，通过对 $F(z^{-1})$ 中参数的调整，可使特征方程的根位于单位圆内。

② 由图 7-5 得，在干扰 $d(k)$ 作用下内模控制反馈为

$$\tilde{d}(k)=\frac{F(z^{-1})}{1+G_c(z^{-1})F(z^{-1})[G(z^{-1})-\tilde{G}(z^{-1})]}d(k) \tag{7-42}$$

当$G(z^{-1})=\tilde{G}(z^{-1})$时，式（7-42）变为

$$\tilde{d}(k)=F(z^{-1})d(k) \tag{7-43}$$

由此可知，$\tilde{d}(k)$包含着模型失配的信息，改变$\tilde{d}(k)$的特性，就能获得不同的鲁棒性，而改变$\tilde{d}(k)$的特性是由选择不同的$F(z^{-1})$来实现的。

由此可见，滤波器$F(z^{-1})$在这里是一个关键因素，$F(z^{-1})$中的参数能直接对系统的鲁棒性进行调整。若有模型失配，则$F(z^{-1})$的作用将使失配的影响减少到尽量小的程度，而且还能镇定由于模型失配可能带来的不稳定性；若无模型失配，则$F(z^{-1})$又可补偿一类动态干扰。若存在模型失配，由式（7-43）直观地可知，要减小失配的影响，可使$F(z^{-1})$减小；而由式（7-35）知，增加柔化因子可使$F(z^{-1})$减小，故增大柔化作用（增大α），可提高系统的鲁棒性。

鲁棒性是预测控制突出的优点之一，现有的文献表明，预测控制在工业过程控制中获得成功的应用，首先应归功于预测控制的鲁棒性。

7.3.2 广义预测控制与动态矩阵控制规律的等价性证明

1．GPC 的最优控制律

由 7.2 节可知，系统采用 CARIMA 模型式（7-17）

$$A(z^{-1})y(k)=B(z^{-1})u(k-1)+C(z^{-1})\xi(k)/\Delta$$

对目标函数式（7-18）

$$J=\sum_{j=1}^{n}[y(k+j)-w(k+j)]^2+\sum_{j=1}^{m}\lambda(j)[\Delta u(k+j-1)]^2$$

GPC 的最优控制律为式（7-27）

$$\Delta U=(G^{\mathrm{T}}G+\lambda I)^{-1}G^{\mathrm{T}}(W-f)$$

式中，$\Delta U=[\Delta u(k),\Delta u(k+1),\cdots,\Delta u(k+n-1)]^{\mathrm{T}}$

$W=[w(k+1),w(k+2),\cdots,w(k+n)]^{\mathrm{T}}$

$f=[f(k+1),f(k+2),\cdots,f(k+n)]^{\mathrm{T}}$

$$G=\begin{bmatrix} g_0 & & & 0 \\ g_1 & g_0 & & \\ \vdots & \vdots & \ddots & \\ g_{n-1} & g_{n-2} & \cdots & g_0 \end{bmatrix}$$

矩阵G中的元素$g_0,g_1,\cdots,g_{n-1}$为系统单位阶跃响应的前n项。

2. DMC 的最优控制律

由 7.1 节可知，若已知对象的单位阶跃响应 $g_0, g_1, \cdots, g_{n-1}, \cdots$，则对应式（7-9）

$$J = \sum_{j=1}^{n}[y(k+j) - w(k+j)]^2 + \sum_{j=1}^{m}\lambda(j)[\Delta u(k+j-1)]^2$$

所表示的目标函数。

DMC 的最优控制律为式（7-11）

$$\Delta \boldsymbol{U} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \lambda \boldsymbol{I})^{-1}\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{W} - \boldsymbol{Y}_0)$$

式中，$\Delta \boldsymbol{U} = [\Delta u(k), \Delta u(k+1), \cdots, \Delta u(k+n-1)]^{\mathrm{T}}$

$\boldsymbol{W} = [w(k+1), w(k+2), \cdots, w(k+n)]^{\mathrm{T}}$

$\boldsymbol{Y}_0 = [y_0(k+1), y_0(k+2), \cdots, y_0(k+n)]^{\mathrm{T}}$

$$\boldsymbol{A} = \begin{bmatrix} a_1 & & & \boldsymbol{0} \\ a_2 & a_1 & & \\ \vdots & \vdots & \ddots & \\ a_n & a_{n-1} & \cdots & a_1 \end{bmatrix} = \begin{bmatrix} g_0 & & & \boldsymbol{0} \\ g_1 & g_0 & & \\ \vdots & \vdots & \ddots & \\ g_{n-1} & g_{n-2} & \cdots & g_0 \end{bmatrix}$$

矩阵 $\boldsymbol{A}$ 中的元素 $a_1, a_2, \cdots, a_n$ 为系统单位阶跃响应的前 n 项。

3. GPC 与 DMC 的等价性

由上可得如下结论：

（1）GPC 中 $\boldsymbol{G}$ 矩阵和 DMC 中 $\boldsymbol{A}$ 矩阵中的元素同为系统单位阶跃响应的前 n 项，故对同一系统应有 $\boldsymbol{A}=\boldsymbol{G}$。

（2）GPC 中的$\Delta \boldsymbol{U}$ 与 DMC 中的$\Delta \boldsymbol{U}$ 完全一样，同为控制量序列，即

$$\Delta \boldsymbol{U} = [\Delta u(k), \Delta u(k+1), \cdots, \Delta u(k+n-1)]^{\mathrm{T}}$$

（3）GPC 中的 $\boldsymbol{W}$ 与 DMC 中的 $\boldsymbol{W}$ 一样，同为柔化作用后的输出跟踪序列，即

$$\boldsymbol{W} = [w(k+1), w(k+2), \cdots, w(k+n)]^{\mathrm{T}}$$

$$w(k+j) = \alpha^j y(k) + (1-\alpha^j)y_r \qquad (j = 1, 2, \cdots, n)$$

（4）式（7-27）的导出是基于由式（7-17）表述的 CARIMA 参数模型，而式（7-11）导出是基于由式（7-17）表述的同一系统的非参数模型。众所周知，对同一线性系统，在同一目标函数下最优控制解具有惟一性。因式（7-18）与式（7-9）相同，故式（7-27）和式（7-11）完全等价。

（5）因 GPC 的控制规律与 DMC 的控制规律完全等价，故 GPC 中的 $\boldsymbol{f}$ 向量相当于 DMC 中的 $\boldsymbol{Y}_0$ 向量，而 $\boldsymbol{Y}_0$ 的物理意义是明确的，它是在 k 时刻预测的未来 n 个时刻未加控制增量$\Delta u(k)$的系统输出量。从而可知，$\boldsymbol{f}$ 就表示 k 时刻基于以往数据对未来输出的预测。

7.3.3 广义预测控制与动态矩阵控制的比较

广义预测控制汲取了动态矩阵控制的多步预测和滚动优化策略，同时又保持了自校正控制器的优点。把它与动态矩阵控制作一比较，对于理解其控制机理及优良性能是十分有益的。

1. 预测模型

GPC 采用了 CARIMA 模型，可用来描述包括不稳定系统在内的任意对象，而 DMC 采用有限卷积模型作为预测模型，只适用于渐近稳定的对象，若不加以修改，适用范围是有局限性的。

2. 控制机理

由于优化策略的一致，两者的控制机理是相同的，GPC 系统的动态响应不仅取决于对象参数及优化设计参数，同时也取决于参考轨迹。

3. 反馈校正

在模型失配或存在干扰时，GPC 和 DMC 都可以通过滤波器 $F(z^{-1})$抑制干扰或保持闭环稳定性。DMC 采用了误差校正修正预测值的策略，主要是通过选择误差校正系数，或者增加滤波器的零点抑制干扰，或者增加其极点改善鲁棒性，但因两者的设计用了同一设计参数，往往难以兼顾。而在 GPC 中，通过增加滤波器的零点抑制干扰。而对模型失配，主要是通过模型在线辨识和自校正来纠正的。因此，GPC 采用了不同的反馈机制分别对付干扰和模型失配，综合的控制效果会更好些。

由此可见，广义预测控制汲取了动态矩阵控制的优化策略，而在预测模型、反馈机制方面都保留了自校正控制的优点，它依靠多步预测及滚动优化获取良好的动态性能，利用在线辨识与校正增强控制系统的鲁棒性，以反馈环节有力地抑制了干扰。

第 8 章 MATLAB 预测控制工具箱函数

MATLAB 的模型预测控制工具箱提供了一系列用于模型预测控制的分析、设计和仿真的函数。这些函数的类型主要有：

（1）系统模型辨识函数——主要功能包括通过多变量线性回归方法计算 MISO 脉冲响应模型和阶跃响应模型及对测量数据的归一化等；

（2）模型建立和转换函数——主要功能包括建立模型预测控制工具箱使用的 MPC 状态空间模型及状态空间模型与 MPC 状态空间模型、阶跃响应模型、脉冲响应模型之间的转换；

（3）模型预测控制器设计和仿真工具主要功能包括面向阶跃响应模型的预测控制器设计与仿真函数和面向 MPC 状态空间模型的设计和仿真函数两类；

（4）系统分析工具主要功能包括计算模型预测控制系统频率响应、极点和奇异值的有关函数；

（5）其他功能函数主要功能包括绘图和矩阵计算函数等。

本章将对模型预测控制工具箱的主要函数的原理和使用方法按以上分类进行介绍，并给出若干设计应用例子。

8.1 系统模型辨识函数

为进行模型预测控制器设计，需要根据系统的输入/输出数据建立系统的脉冲响应模型或阶跃响应模型，即进行系统模型的辨识。MATLAB 模型预测控制工具箱提供的模型辨识函数见表 8-1。

表 8-1 系统模型辨识函数

函 数 名	功 能
autosc( )	矩阵或向量的自动归一化
scal( )	根据指定的均值和标准差归一化矩阵
rescal( )	由归一化的数据生成源数据
wrtreg( )	生成用于线性回归计算的数据矩阵
mlr( )	利用多变量线性回归计算 MISO 脉冲响应模型
plsr( )	利用部分最小二乘回归方法计算 MISO 脉冲响应模型
imp2step( )	由 MISO 脉冲响应模型生成 MIMO 阶跃响应模型
validmod( )	利用新的数据检验 MISO 脉冲响应模型

8.1.1 数据向量或矩阵的归一化

在获得系统输入/输出的原始数据后，为进行参数估计和模型辨识，往往要求对数据进行归一化处理。模型预测控制工具箱提供的有关数据归一化处理的函数有：autosc( )、scal( )和 rescal( )，这三个函数都是根据数据的均值和标准差来对数据进行归一化或反归一化处理的。设有数据向量为

$$\boldsymbol{x}=[x_1,x_2,\cdots,x_n]^{\mathrm{T}}$$

其均值和标准差分别为

$$\bar{x}=\frac{1}{n}\sum_{i=1}^{n}x_i,\sigma=\sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}}$$

进行归一化处理后得到的向量为

$$\boldsymbol{x}=\left[\frac{x_1-\bar{x}}{\sigma},\frac{x_2-\bar{x}}{\sigma},\cdots,\frac{x_n-\bar{x}}{\sigma}\right]^{\mathrm{T}}$$

1．矩阵或向量的自动归一化函数 autosc( )

该函数的调用格式为

[ax,mx,stdx]=autosc(x)

式中，x 为数据向量或矩阵；ax 为归一化后的向量或矩阵，若输入参数 x 为矩阵，则对其每一列进行归一化计算；mx 为均值向量，当输入参数 x 为矩阵时，mx 为 x 的各列向量的均值构成的向量；stdx 为标准差向量，当输入参数 x 为矩阵时，stdx 为 x 的各列向量的标准差构成的向量，如

```
>>x=[1  1  1  2];[ax,mx,stdx]=autosc(x)
```

其输出结果为

```
ax =
        -0.5000    -0.5000    -0.5000     1.5000
mx =
        1.2500
stdx =
        0.5000
```

2．根据指定的均值和标准差进行矩阵或向量的归一化函数 scal( )

该函数的调用格式为

sx=scal(x,mx)

sx=scal(x,mx,stdx)

式中，x 为数据向量或矩阵；mx 为指定的均值向量，当输入参数 x 为矩阵时，向量 mx 用

于指定对 x 的各列向量进行归一化的均值；stdx 为指定的标准差向量，当输入参数 x 为矩阵时，stdx 用于指定对 x 的各列向量进行归一化的标准差；sx 为归一化后的向量或矩阵，若输入参数 x 为矩阵，则对其每一列进行归一化计算，如以下 MATLAB 命令：

```
>>x=[1  2; 2  1]; sx=scal(x,[1  1],[0.5  0.5])
```

其输出结果为

```
sx =
        0    2
        2    0
```

3．由归一化的矩阵或向量计算原矩阵或向量（反归一化）函数 rescal( )

该函数的调用格式为

rx=rescal(x,mx)

rx=rescal(x,mx,stdx)

式中，x 为数据向量或矩阵；mx 为用于反归一化的均值向量；stdx 为用于反归一化的标准差向量；rx 为反归一化得到的矩阵或向量，如以下 MATLAB 命令：

```
>>x=[0  2;2  0]; rx=rescal(x,[1  1],[0.5  0.5])
```

其输出结果为

```
rx =
    1    2
    2    1
```

8.1.2　基于线性回归方法的脉冲响应模型辨识

在获得系统的输入/输出数据之后，可以采用最小二乘法或部分最小二乘法等线性回归方法来计算系统的脉冲响应模型的系数。在进行线性回归计算之前，需要对原始数据进行预处理，函数 wrtreg( )用于完成这一预处理过程。函数 mlr( )和 plsr( )分别完成基于多变量最小二乘法和部分最小二乘法的脉冲响应模型辨识。

1．生成用于线性回归计算的输入/输出数据矩阵函数 wrtreg( )

该函数的调用格式为

[xreg,yreg]=wrtreg(x,y,n)

式中，x 为输入数据矩阵；y 为输出数据向量；n 为脉冲响应模型系数的个数；xreg 为预处理后的输入数据矩阵；yreg 为预处理后的输出数据向量。

2．基于多变量最小二乘法的脉冲响应模型辨识函数 mlr( )

该函数的调用格式为

[theta,yres]=mlr(xreg,yreg,ninput)

[theta,yres]=mlr(xreg,yreg,ninput,plotopt,wtheta,wdeltheta)

式中，xreg 为预处理后的输入数据矩阵；yreg 为预处理后的输出数据向量；ninput 为输入变量的个数；plotopt 为绘图选项：plotopt=0，默认值，不绘制图形；plotopt=1，绘制实际输出和预测输出；plotopt=2，绘制实际输出、预测输出及输出误差；wtheta 和 wdeltheta 为最小二乘法的加权向量，默认值均为 0；theta 为脉冲响应模型的系数矩阵，该矩阵的每一列对应一个输入到输出的脉冲响应模型系数向量；yres 为预测误差向量。

例 8-1 考虑一个单输入单输出的对象，其传递函数为

$$G(s)=\frac{s+1}{s^2+3s+6}$$

解：采用下面的 MATLAB 程序对该对象进行脉冲响应模型辨识，脉冲响应模型预测输出与预测误差曲线如图 8-1 所示。

```
%将多项式的传递函数模型转换为 MPC 传递函数模型
num=[1  1];den=[1  3  6];
h=tf(num,den);
%获得脉冲信号 x
[u,t]=gensig('pulse',2,10,0.1);
x=u;
%求解 LTI 对象的单位脉冲响应 y
t=0:0.1:10;
[y,x1,t1]=lsim(h,x,t);
%输入脉冲信号 x 的归一化处理
[ax,mx,stdx]=autosc(x);
mx=0;
sx=scal(x,mx,stdx);
%生成用于线性回归计算的输入/输出数据矩阵
n=35;
[xreg,yreg]=wrtreg(sx,y,n);
%基于多变量最小二乘法的脉冲响应模型辨识
ninput=1;
plotopt=2;
[theta,yres]=mlr(xreg,yreg,ninput,plotopt)
```

例 8-2 考虑一个双输入单输出的对象，其传递函数矩阵为

$$G(s)=\begin{bmatrix}\frac{5.72e^{-14s}}{60s+1} & \frac{1.52e^{-15s}}{25s+1}\end{bmatrix}$$

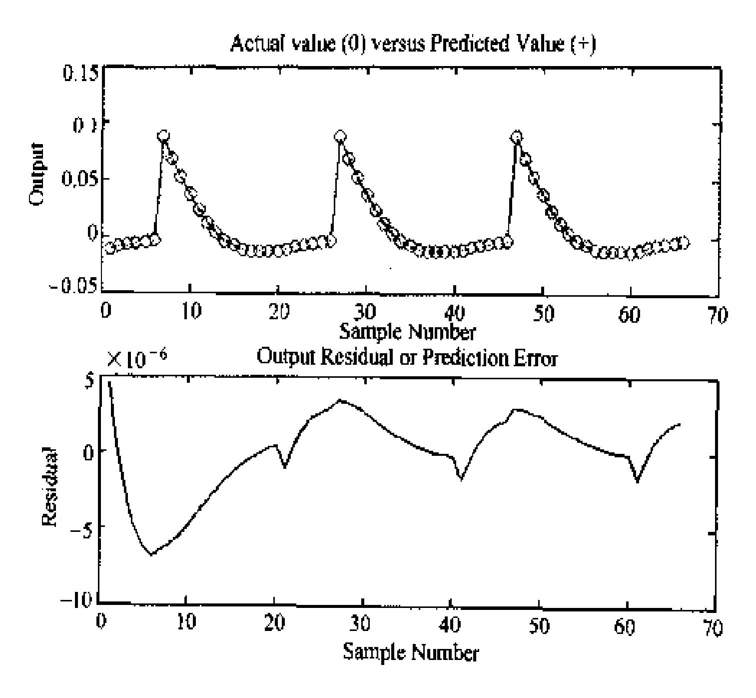


图 8-1 脉冲响应模型预测输出与预测误差曲线

采样时间为 7s。

解：采用下面的 MATLAB 程序对该对象进行脉冲响应模型辨识，脉冲响应模型预测输出与预测误差曲线如图 8-2 所示。

```
%将多项式的传递函数模型转换为 MPC 传递函数模型
num1=5.72;den1=[60 1];g1=poly2tfd(num1,den1,0,14);
num2=1.52;den2=[25 1];g2=poly2tfd(num2,den2,0,15);
%将 MPC 传递函数模型转换为 MPC 状态空间模型
mod=tfd2mod(7,1,g1,g2);
%将 MPC 状态空间模型转换为通用状态空间模型
[A,B,C,D]=mod2ss(mod);
%将通用状态空间模型转换 LTI 对象的状态空间模型
sys=ss(A,B,C,D);
h=tf(sys);
%获得脉冲信号 x
[u,t]=gensig('pulse',8,10,0.1);
x=[u u];
%求解 LTI 对象的单位脉冲响应 y
t=0:0.1:10;
[y,x1,t1]=lsim(h,x,t);
%输入脉冲信号 x 的归一化处理
```

```
[ax,mx,stdx]=autosc(x);
mx=[0 0];
sx=scal(x,mx,stdx);
%生成用于线性回归计算的输入/输出数据矩阵
n=35;
[xreg,yreg]=wrtreg(sx,y,n);
%基于多变量最小二乘法的脉冲响应模型辨识
ninput=2;
plotopt=2;
[theta,yres]=mlr(xreg,yreg,ninput,plotopt)
```

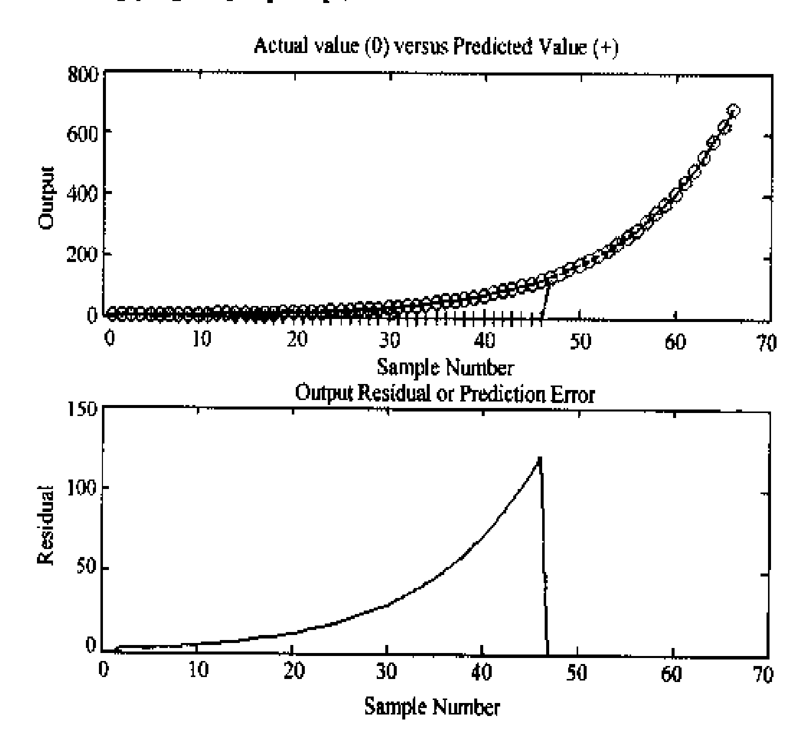


图 8-2 脉冲响应模型预测输出与预测误差曲线

3. 基于部分最小二乘法（PLS）的脉冲响应模型辨识函数 plsr( )

该函数的调用格式为

[theta,w,cw,ssqdif,yres]=plsr(xreg,yreg,ninput,lv,plotopt)

式中，xreg 为预处理后的输入数据矩阵；yreg 为预处理后的输出数据向量；ninput 为输入变量的个数；plotopt 为绘图选项：plotopt＝0 为默认值，不绘制图形；plotopt=l，绘制实际输出和预测输出；plotopt=2，绘制实际输出；lv 为用于指定最大化输入/输出协方差的方向的数目；w,cw,ssqdif 为部分最小二乘法的有关计算结果；theta 为脉冲响应模型的系数矩阵；yres 为预测输出误差。

例 8-3　考虑一个单输入单输出的对象，其传递函数为

$$G(s)=\frac{s+1}{s^2+3s+6}$$

解：采用如下的 MATLAB 程序对该系统进行模型响应模型辨识，其模型的预测输出和预测误差曲线如图 8-3 所示。

```
%将多项式的传递函数模型转换为 MPC 传递函数模型
num=[1  1];den=[1  3  6]; h=tf(num,den);
%获得脉冲信号 x
[u,t]=gensig('pulse',2,10,0.1);x=[u];
%求解 LTI 对象的单位脉冲响应 y
t=0:0.1:10; [y,x1,t1]=lsim(h,x,t);
%生成用于线性回归计算的输入/输出数据矩阵
n=30; [xreg,yreg]=wrtreg(x,y,n);
%基于部分最小二乘法（PLS）的脉冲响应模型辨识
ninput=2;lv=10;plotopt=2;
theta=plsr(xreg,yreg,ninput,lv,plotopt)
```

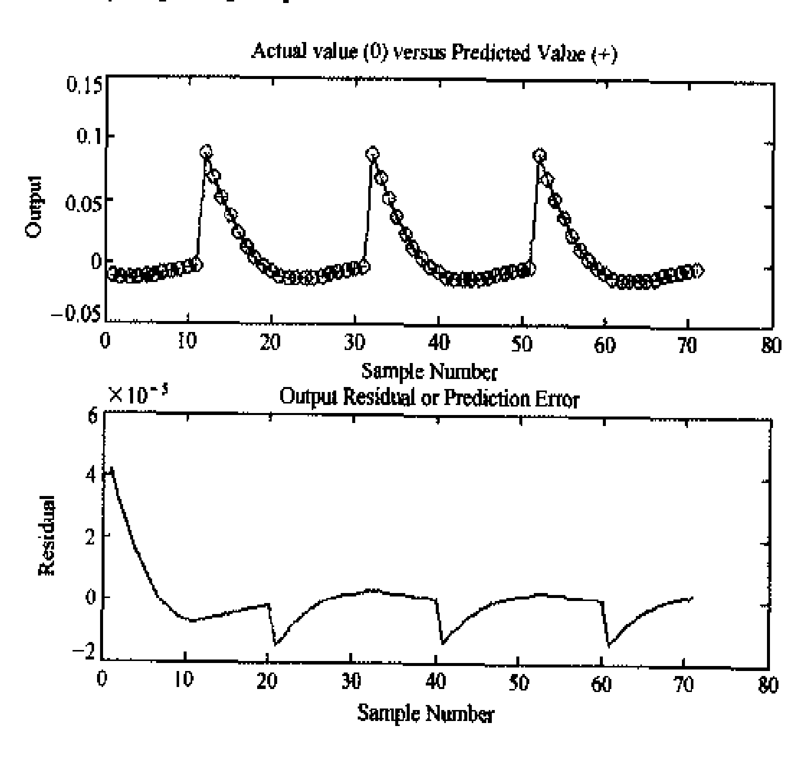


图 8-3　基于 PLS 线性回归的模型预测输出与预测误差曲线

8.1.3 脉冲响应模型转换为阶跃响应模型

在模型预测控制工具箱中，模型预测控制器的设计主要基于系统阶跃响应模型。因此

在完成系统的脉冲响应模型辨识后，往往需要转换为等价的阶跃响应模型。函数 imp2step ( ) 用于完成这一转换过程，该函数的调用格式为

plant=imp2step(delt,nout,theta1,theta2,…)

式中，delt 为脉冲响应模型的采样周期；nout 为用于指定输出的稳定性，对于稳定的系统，nout 等于输出的个数；对于具有一个或多个积分输出的系统，nout 为一个长度等于输出个数的向量，该向量对应积分输出的分量为 0，其余分量为 1；theta1，theta2，…为脉冲响应系数矩阵，其中 thetai 对应第 i 个输出，i 的最大取值为 25；plant 为系统的阶跃响应系数矩阵，维数为(n×ny+ny+2)×nu，其中，n 为对应每个输入的系数个数，nu 和 ny 分别为输入和输出变量个数。

例 8-4 考虑一个单输入单输出的对象，其传递函数为

$$G(s)=\frac{s+1}{s^2+3s+6}$$

解：采用如下的 MATLAB 语句对该系统进行脉冲响应模型辨识，并转换为阶跃响应模型，系统的阶跃响应的预测曲线如图 8-4 所示。

```
%将多项式的传递函数模型转换为 MPC 传递函数模型
num=[1  1];den=[1  3  6]; h=tf(num,den);
%获得脉冲信号 x
[u,t]=gensig('pulse',2,10,0.1); x=[u];
%求解 LTI 对象的单位脉冲响应 y
t=0:0.1:10;
[y,x1,t1]=lsim(h,x,t);
%输入脉冲信号 x 的归一化处理
[ax,mx,stdx]=autosc(x);mx=[0];
sx=scal(x,mx,stdx);
%生成用于线性回归计算的输入/输出数据矩阵
n=35;
[xreg,yreg]=wrtreg(sx,y,n);
%基于多变量最小二乘法的脉冲响应模型辨识
ninput=1; plotopt=2;
[theta,yreg]=mlr(xreg,yreg,ninput,plotopt);
%脉冲响应模型转换为阶跃响应模型
theta=scal(theta,mx,stdx);
nout=1; delt=1;
model=imp2step(delt,nout,theta);
```

plotstep(model)

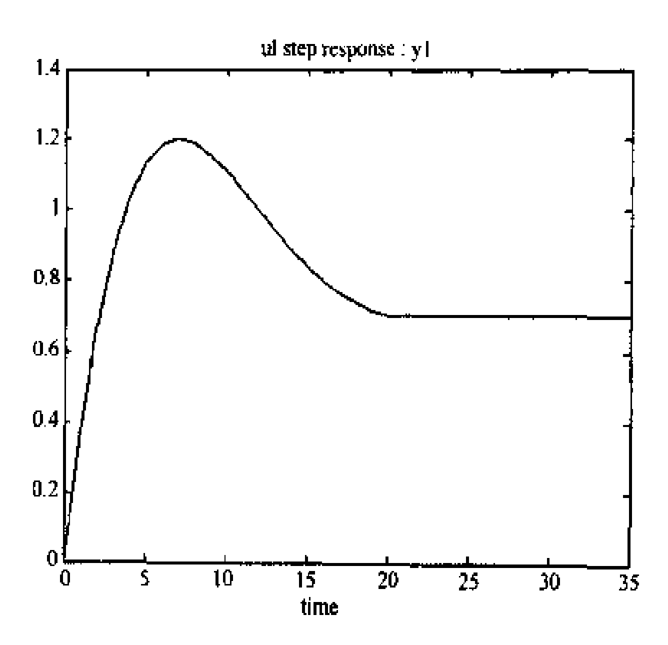


图 8-4　系统阶跃响应的预测曲线

8.1.4　模型的校验

在根据系统的输入/输出数据完成模型辨识以后，可以进一步利用新的数据对辨识模型进行校验，函数 validmod( )用于实现这一功能。该函数的调用格式为

yres=validmod(xreg,yreg,theta,plotopt)

式中，xreg 为经过预处理的输入校验数据；yreg 为经过预处理的输出校验数据；theta 为脉冲响应模型的系数矩阵；plotopt 为绘图选项：plotopt=0 为默认值，不绘制图形；plotopt=1 为绘制实际输出和预测输出；plotopt=2 为绘制实际输出；yres 为输出预测误差。

8.2　系统模型建立与转换函数

前面讨论了利用系统输入/输出数据进行系统模型辨识的有关函数及使用方法，为进行模型预测控制器的设计，需要对系统模型进行进一步的处理和转换。MATLAB 的模型预测控制工具箱中提供了一系列函数完成多种模型转换和复杂系统模型的建立功能。在模型预测控制工具箱中使用了两种专用的系统模型格式，即 MPC 状态空间模型和 MPC 传递函数模型。这两种模型格式分别是状态空间模型和传递函数模型在模型预测控制工具箱中的特殊表达形式。这两种模型格式可以同时支持连续和离散系统模型的表达，在 MPC 传递函数模型中还增加了对纯迟延的支持。表 8-2 列出了模型预测控制工具箱的模型建立与转换函数。

表 8-2 模型建立与转换函数

函 数 名	功 能
ss2mod( )	将通用状态空间模型转换为 MPC 状态空间模型
mod2ss( )	将 MPC 状态空间模型转换为通用状态空间模型
poly2tfd( )	将通用传递函数模型转换为 MPC 传递函数模型
tfd2mod( )	将 MPC 传递函数模型转换为 MPC 状态空间模型
mod2step( )	将 MPC 状态空间模型转换为 MPC 阶跃响应模型
tfd2step( )	将 MPC 传递函数模型转换为 MPC 阶跃响应模型
ss2step( )	将通用状态空间模型转换为 MPC 阶跃响应模型
mod2mod( )	改变 MPC 状态空间模型的采样周期
th2mod( )	将 Theta 格式模型转换为 MPC 状态空间模型
addmod( )	将两个开环 MPC 模型连接构成闭环模型，使其中一个模型输出叠加到另一个模型输入
addmd( )	向 MPC 对象添加一个或多个测量扰动
addumd( )	向 MPC 对象添加一个或多个未测量扰动
paramod( )	将两个 MPC 系统模型并联
sermod( )	将两个 MPC 系统模型串联
appmod( )	用两个 MPC 系统模型构成增广系统模型

8.2.1 模型转换

在 MATLAB 模型预测控制工具箱中支持多种系统模型格式。这些模型格式包括：

① 通用状态空间模型；

② 通用传递函数模型；

③ MPC 阶跃响应模型；

④ MPC 状态空间模型；

⑤ MPC 传递函数模型。

在上述模型格式中，前两种模型格式是 MATLAB 通用的模型格式，在其他控制类工具箱中，如控制系统工具箱、鲁棒控制工具箱等都予以支持；而后三种模型格式则是模型预测控制工具箱特有的。其中，MPC 状态空间模型和 MPC 传递函数模型是通用的状态空间模型和传递函数模型在模型预测控制工具箱中采用的增广格式，为了方便计算。模型预测控制工具箱中提供了若干函数，用于完成上述模型格式之间的转换功能。下面对这些函数的用法加以介绍。

1. 通用状态空间模型与 MPC 状态空间模型之间的转换

MPC 状态空间模型在通用状态空间模型的基础上增加了对系统输入/输出扰动和采样周期的描述信息，函数 ss2mod( )和 mod2ss( )用于实现这两种模型格式之间的转换。

1）通用状态空间模型转换为 MPC 状态空间模型函数 ss2mod( )

该函数的调用格式为

pmod=ss2mod(A,B,C,D)

pmod=ss2mod(A,B,C,D,minfo)

pmod=ss2mod(A,B,C,D,minfo,x0,u0,y0,f0)

式中，A,B,C,D 为通用状态空间矩阵；minfo 为构成 MPC 状态空间模型的其他描述信息，为 7 个元素的向量，各元素分别定义为：minfo(1)=dt，系统采样周期，默认值为 1；minfo(2)=n，系统阶次，默认值为矩阵 A 的阶次；minfo(3)=nu，受控输入的个数，默认值为系统输入的维数；minfo(4)=nd，测量扰动的数目，默认值为 0；minfo(5)=nw，未测量扰动的数目，默认值为 0；minfo(6)=nym，测量输出的数目，默认值为系统输出的维数；minfo(7)=nyu，未测量输出的数目，默认值为 0；x0,u0,y0,f0 为线性化条件，默认值均为 0；如果在输入参数中没有指定 minfo，则取默认值；pmod 为系统的 MPC 状态空间模型格式。

例 8-5　将如下以传递函数表示的系统模型转换为 MPC 状态空间模型。

$$G(s)=\frac{s^2+3s+1}{s^3+2s^2+2s+1}$$

解：MATLAB 命令如下：

```
>>num=[1 3 1];den=[1 2 2 1];[A,B,C,D]=tf2ss(num,den)
>>pmod=ss2mod(A,B,C,D)
```

其输出结果为

```
A =
    -2    -2    -1
     1     0     0
     0     1     0
B =
     1
     0
     0
C =
     1     3     1
D =
     0
pmod =
     1     3     1     0     0     1     0
   NaN    -2    -2    -1     1     0     0
```

```
0    1    0    0    0    0    0
0    0    1    0    0    0    0
0    1    3    1    0    0    0
```

2）MPC 状态空间模型转换为通用状态空间模型函数 mod2ss( )

该函数的调用格式为

```
[A,B,C,D]=mod2ss(pmod)
[A,B,C,D,minfo]=mod2ss(pmod)
[A,B,C,D,minfo,x0,u0,y0,f0]= mod2ss(pmod)
```

式中，pmod 为系统的 MPC 状态空间模型格式；A,B,C,D 为通用状态空间矩阵；minfo 为构成 MPC 状态空间模型的其他描述信息，其说明参见函数 ss2mod( )。

2. 通用传递函数模型转换为 MPC 传递函数模型

通用传递函数模型与 MPC 传递函数模型的转换函数 ploy2tfd( ) 的调用格式为

```
g=ploy2tfd(num,den,delt,delay)
```

式中，num 为通用传递函数模型的分子多项式系数向量；den 为通用传递函数模型的分母多项式系数向量；delt 为采样周期，对连续系统，该参数为 0；delay 为系统纯时延，对于离散系统，纯时延为采样周期的整数倍；g 为对象的 MPC 传递函数模型。

例 8-6 考虑如下的纯时延二阶对象，并将其转换为 MPC 传递函数模型。

$$G(s)=\frac{e^{-0.5s}(s+1)}{s^2+4s+4}$$

解：MATLAB 命令如下：

```
>>num=[1  1];den=[1  4  4];
>>g=poly2tfd(num,den,0,0.5);
```

结果显示

```
g =
         0    1.0000    1.0000
    1.0000    4.0000    4.0000
         0    0.5000         0
```

3. MPC 传递函数模型转换为 MPC 状态空间模型函数 tfd2mod( )

该函数的调用格式为

```
pmod=tfd2mod(delt,ny,g1,g2,g3,…,g25)
```

式中，delt 为采样周期；ny 为输出个数；g1，g2，…为 SISO 传递函数，对应多变量系统传递函数矩阵的各个元素按行向量顺序排列构成的向量，其最大个数限制为 25；pmod 为系统的 MPC 状态空间模型。

例 8-7 考虑如下的二阶对象，并将其转换为 MPC 状态空间模型。

$$G(s)=\frac{s+1}{s^2+3s+6}$$

解：MATLAB 命令如下：

```
>>num=[1   1];den=[1   3   6];
>>g=poly2tfd(num,den,0,0)
>>mod1=tfd2mod(0.1,1,g)
```

结果显示

```
  g =
      0     1     1
      1     3     6
      0     0     0
mod1 =
   0.1000    2.0000    1.0000         0         0    1.0000         0
      NaN    1.6892   -0.7408    1.0000         0         0         0
        0    1.0000         0         0         0         0         0
        0    0.0900   -0.0815         0         0         0         0
```

4．MPC 阶跃响应模型与其他模型格式之间的转换

函数 mod2step( )、tfd2step( )和 ss2step( )分别用于将 MPC 状态空间模型、MPC 传递函数模型和通用状态空间模型转换为 MPC 阶跃响应模型。下面对这三个函数的用法进行说明。

1）MPC 状态空间模型转换为 MPC 阶跃响应模型函数 mod2step( )

该函数的调用格式为

```
plant=mod2step(pmod,tfinal)
[plant,dplant]=mod2step(pmod,tfinal,delt,nout)
```

式中，pmod 为系统的 MPC 状态空间模型；tfinal 为阶跃响应模型的截断时间；delt 为采样周期，默认值由 MPC 状态空间模型的参数 minfo(1)决定；nout 为输出稳定性向量，用于指定输出的稳定性，对于稳定的系统，nout 等于输出的个数；对于具有一个或多个积分输出的系统，nout 为一个长度等于输出个数的向量，该向量对应积分输出的分量为 0，其余分量为 1；plant 为对象在受控变量作用下的阶跃响应系数矩阵；dplant 为对象在扰动作用下的阶跃响应矩阵。

2）MPC 传递函数模型转换为 MPC 阶跃响应模型函数 tfd2step( )

该函数的调用格式为

```
plant=tfd2step(tfinal,delt,nout,g1)
plant=tfd2step(tfinal,delt,nout,g1,…,g25)
```

式中，tfinal 为阶跃响应的截断时间；delt 为采样周期；nout 为输出稳定性向量，参见函数

mod2step( )的有关说明；g1, g2，…为 SISO 传递函数，对应多变量系统传递函数矩阵的各个元素按行向量顺序排列构成的向量，其最大个数限制为 25；plant 为对象的阶跃响应系数矩阵。

例 8-8 设系统传递函数为

$$G(s)=\frac{s+2}{s^2+3s+1}$$

将其转换为阶跃响应模型。

解：MATLAB 命令如下：

```
>>num=[1  2];den=[1  3  1];
>>tf1=poly2tfd(num,den,0,0);
>>plant=tfd2step(5,0.1,1,tf1);
>>plotstep(plant)
```

由阶跃响应模型绘制的系统阶跃响应曲线如图 8-5 所示。

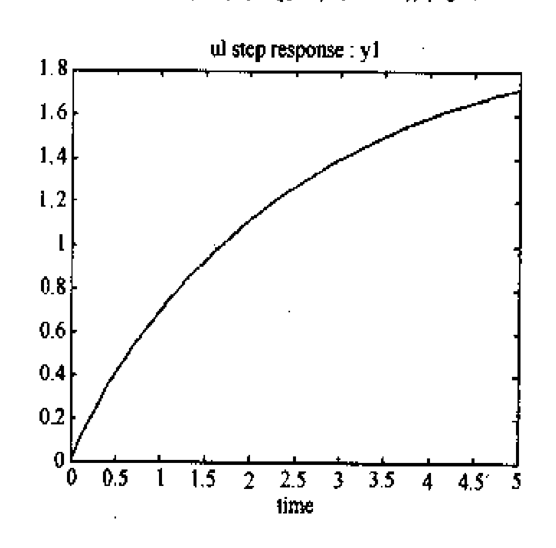


图 8-5 系统阶跃响应曲线

例 8-9 考虑如下的双输入双输出系统的传递函数矩阵

$$G(s)=\begin{bmatrix}\dfrac{2}{s+1} & \dfrac{3}{s+2}\\ \dfrac{1}{s+4} & \dfrac{1}{s+1}\end{bmatrix}$$

将多变量系统传递函数模型转换为阶跃响应模型。

解：MATLAB 命令如下：

```
>>tf1=poly2tfd(2,[1,1],0,0);
>>tf2=poly2tfd(3,[1 2],0,0);
>>tf3=poly2tfd(1,[1,4],0,0);
>>tf4=poly2tfd(1,[1,1],0,0);
```

```
>>plant=tfd2step(5,0.1,2,tf1,tf2,tf3,tf4);
>>plotstep(plant)
```

系统在阶跃输入作用下的响应曲线如图 8-6 所示。

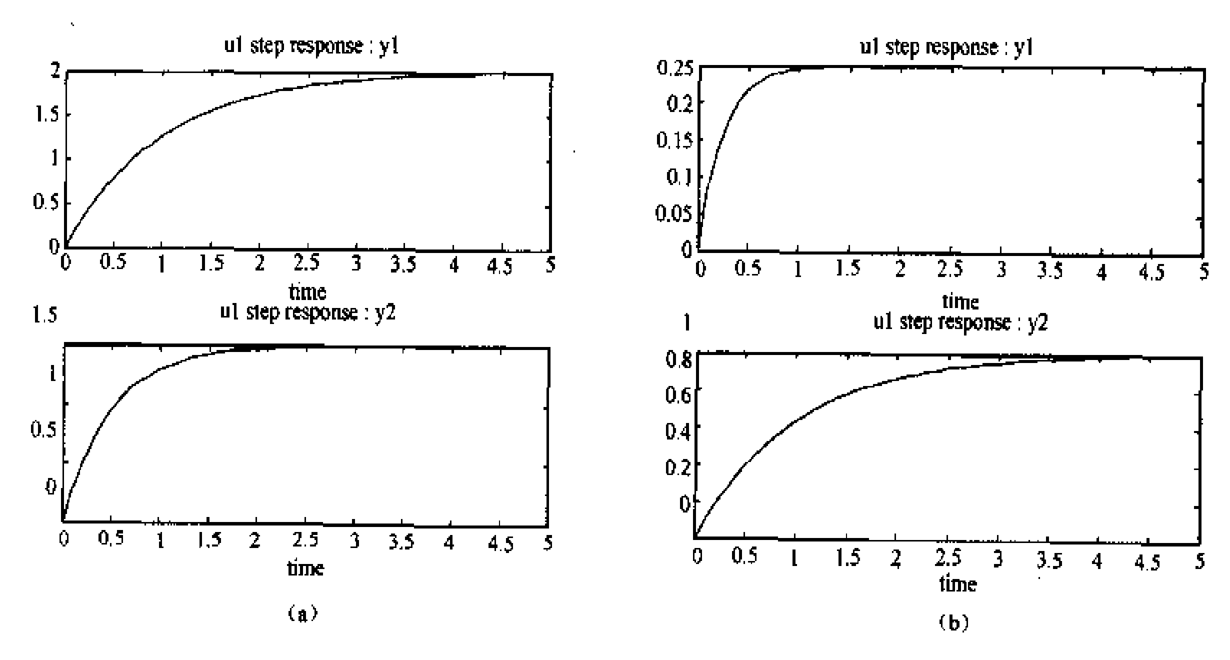


图 8-6　多变量系统的阶跃响应曲线

3）通用状态空间模型转换为 MPC 阶跃响应模型函数 ss2step( )

该函数的调用格式为

plant=ss2step(A,B,C,D,tfinal)

plant=ss2step(A,B,C,D,tfinal,delt1,delt2,nout)

式中，A,B,C,D 为状态空间矩阵；tfinal 为阶跃响应的截断时间；delt1 为系统的采样周期，对于连续系统，该参数为 0；delt2 为阶跃响应模型的采样周期，当指定 delt1 时，默认值为 delt1；当未指定 deltl 时，默认值为 1；nout 为输出稳定性向量，参见函数 mod2step( )的有关说明；plant 为对象的阶跃响应系数矩阵。

例 8-10　考虑如下的系统状态空间模型

$$\boldsymbol{A}=\begin{bmatrix}-2 & -1\\ 1 & 0\end{bmatrix},\boldsymbol{B}=\begin{bmatrix}1\\0\end{bmatrix}$$
$$\boldsymbol{C}=\begin{bmatrix}1 & 2\end{bmatrix},D=0$$

将该模型转换为阶跃响应模型。

解：利用以下 MATLAB 命令可绘制如图 8-7 所示的系统阶跃响应曲线，截断时间为 5s，阶跃响应的采样周期为 0.2。

```
>>A=[-2 -1;1 0];B=[1;0];C=[1 2];D=0;
>>plant=ss2step(A,B,C,D,5,0,0.2,1);
```



```
>>plotstep(plant)
```

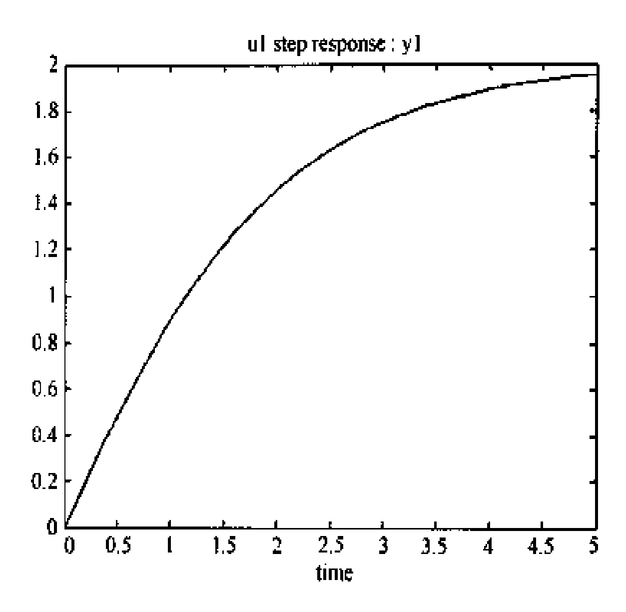


图 8-7 系统阶跃响应曲线

5．其他的模型转换函数

1）改变 MPC 状态空间模型的采样周期函数 mod2mod( )

该函数的调用格式为

newmod=mod2mod(oldmod,delt)

式中，oldmod 为离散系统原有的 MPC 状态空间模型；delt 为指定系统新的采样周期；newmod 为改变采样周期后的系统 MPC 状态空间模型。

例 8-11 考虑二阶对象

$$G(s)=\frac{s+1}{s^2+3s+6}$$

解：下面的 MATLAB 程序用于计算并绘制采样周期为 0.5s 时系统的阶跃响应曲线，如图 8-8 所示。

```
num=[1  1];den=[1  3  6];
%将多项式模型转换为 MPC 传递函数模型
g=poly2tfd(num,den,0,0);
%将 MPC 传递函数模型转换为连续 MPC 状态空间模型
mod1=tfd2mod(0.1,1,g);
mod2=mod2mod(mod1,0.5);          %改变系统的采样周期
plant2=mod2step(mod2,5,0.1);     %计算阶跃响应模型
plotstep(plant2)                 %绘制阶跃响应曲线
```

2）系统辨识工具箱的 Theta 格式模型转换为 MPC mod 格式函数 th2mod( )

该函数的调用格式为

umod=th2mod(th)

[umod,emod]=th2mod(th1,th2,th3,th4,th5,th6,th7,th8)

式中，th 为系统辨识工具箱的 Theta 格式模型，参见系统辨识工具箱；umod 为系统的测量输入对输出影响的 MPC 状态空间模型；emod 为系统的噪声输入对输出影响 MPC mod 的模型。

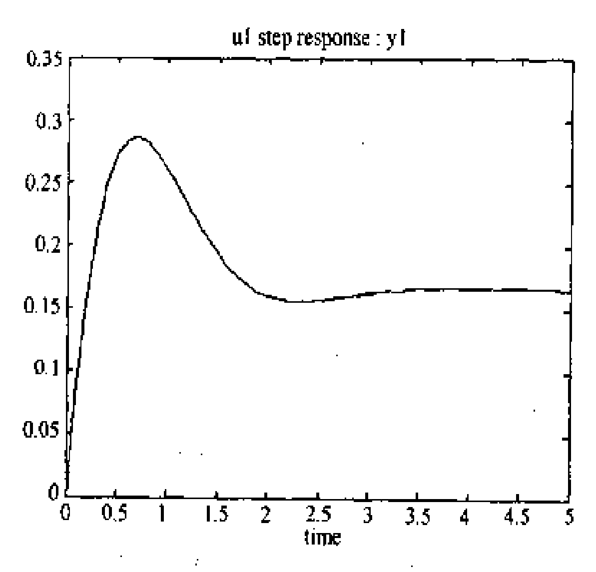


图 8-8　二阶对象的阶跃响应曲线

8.2.2　模型建立

1．计算闭环系统模型函数 addmod( )

在模型建立工具中，函数 addmod ( )用于计算闭环系统的模型。该函数的调用格式为

pmod=addmod(mod1,mod2)

式中，mod1 为前向通道的 MPC 状态空间模型；mod2 为反馈通道的 MPC 状态空间模型；pmod 为闭环系统的 MPC 状态空间模型，如以下 MATLAB 命令：

```
>>num1=[1 1]; den1=[1 3 1];
>>num2=1;den2=[2 1];
>>[A1,B1,C1,D1]=tf2ss(num1,den1);
>>[A2,B2,C2,D2]=tf2ss(num2,den2);
>>mod1=ss2mod(A1,B1,C1,D1);
>>mod2=ss2mod(A2,B2,C2,D2);
>>pmod=addmod(mod1,mod2)
```

结果显示：

```
pmod =
    1.0000    3.0000    2.0000        0        0    1.0000        0
```

```
NaN   -3.0000   -1.0000    0.5000    1.0000         0         0
  0    1.0000         0         0         0         0         0
  0         0         0   -0.5000         0    1.0000         0
  0    1.0000    1.0000         0         0         0         0
```

2. 向对象输出添加测量扰动信号

在实际对象的输出端通常叠加有噪声信号。为进行仿真分析，模型预测控制工具箱的函数 addmd( )和 addumd( )分别用于向对象的输出添加测量扰动信号模型和未测量扰动信号模型。

1）向对象的输出添加测量扰动信号模型函数 addmd( )

该函数的调用格式为：

model=addmd(pmod,dmod)

式中，pmod 为对象的 MPC 状态空间模型；dmod 为测量扰动的 MPC 状态空间模型；model 为叠加了噪声的对象 MPC 状态空间模型。

2）向对象的输出添加未测量扰动信号模型函数 addumd( )

该函数的调用格式为

model=addumd(pmod,dmod)

式中，pmod 为对象的 MPC 状态空间模型；dmod 为未测量扰动的 MPC 状态空间模型。model 为叠加了噪声的对象 MPC 状态空间模型。

3. 系统模型的级联和综合

复杂系统的建模往往要求多个系统模型的级联（包括串联、并联）或将两个模型综合为一个整体系统模型，下面对模型预测控制工具箱的有关函数进行介绍。

1）计算两个系统的并联模型函数 paramod( )

该函数的调用格式为

pmod=paramod(mod1,mod2)

式中，mod1 和 mod2 为两个并联子系统的 MPC 状态空间模型；pmod 为并联后系统的 MPC 状态空间模型。

该系统的输出为两个系统的输出之和，系统的所有输入被重新按类型（包括控制输入、测量扰动和未测量扰动）分组。

2）计算两个系统的串联模型函数 sermod( )

该函数的调用格式为

pmod=sermod(mod1,mod2)

式中，mod1 和 mod2 为两个串联子系统的 MPC 状态空间模型；pmod 为串联后系统的 MPC 状态空间模型，在该模型中，系统 mod1 的测量输出作为系统 mod2 的控制输入，两个子系统的扰动信号仍然作为串联系统的扰动信号。

3）计算两个状态空间模型构成的增广系统模型函数 appmod( )

该函数的调用格式为

pmod=appmod(mod1,mod2)

[pmod,in1,in2,out1,out2]=appmod(mod1,mod2)

式中，mod1 和 mod2 为两个子系统的 MPC 状态空间模型；pmod 为增广系统的 MPC 状态空间模型；in1,in2,out1,out2 为增广系统的输入/输出变量，其中，in1,out1 为由子系统 mod1 输入/输出构成的增广系统输入/输出变量；in2,out2 为由子系统 mod2 输入/输出构成的增广系统输入/输出变量。

8.3 基于阶跃响应模型的控制器设计与仿真函数

基于系统的阶跃响应模型进行模型预测控制器设计的方法称为动态矩阵控制方法。该方法的特点是采用工程上易于获取的对象阶跃响应模型，算法较为简单，计算量较少，鲁棒性较强，适用于纯迟延、开环渐进稳定的非最小相位系统，在工业过程控制中得到成功的应用。

MATLAB 的模型预测控制工具箱提供了对动态矩阵控制方法的支持，有关的函数能够完成基于阶跃响应模型的模型预测控制器设计和仿真，见表 8-3。

表 8-3　动态矩阵控制设计与仿真函数

函数名	功能
cmpc( )	输入/输出有约束的模型预测控制器设计与仿真
mpccon( )	输入/输出无约束的模型预测控制器设计
mpcsim( )	模型预测控制系统的仿真（输入/输出无约束）
mpccl( )	计算模型预测控制系统的闭环模型
nlcmpc( )	Simulink 块 nlcmpc 对应的 S 函数
nlmpcsim( )	Simulink 块 nlmpcsim 对应的 S 函数

8.3.1 输入/输出有约束的模型预测控制器设计与仿真

所谓输入/输出有约束是指系统的输入/输出变量满足一定的上界和下界要求。函数 cmpc( )用于在系统输入/输出变量有约束的情况下进行模型预测控制器的设计和仿真。该函数的调用格式为

[y,u,ym]=cmpc(plant,model,ywt,uwt,M,P, tend,r,ulim,ylim,tfilter,dplant,dmodel,dstep)

式中，plant 为开环对象的实际阶跃响应模型；model 为辨识得到的开环对象阶跃响应模型；ywt 为二次型性能指标的输出误差加权矩阵；uwt 为二次型性能指标的控制量加权矩阵；M 为控制时域长度；P 为预测时域长度，当 P=Inf 时，表示无限的预测和控制时域长度；tend 为仿真的结束时间；r 为输出设定值或参考轨迹。以下的输入参数为可选参数：ulim=[u_{1min}

$u_{2min}\cdots u_{rmin}$ u_{1max} $u_{2max}\cdots u_{rmax}$ Δu_1 Δu_2 $\cdots$ Δu_r] 为输入控制变量的约束矩阵，包括控制变量的下界、上界和变化率曲线的轨迹；ylim=[y_{1min} $y_{2min}\cdots y_{mmin}$ y_{1max} $y_{2max}\cdots y_{mmax}$] 为输出变量的约束矩阵，包括输出变量的下界和上界的轨迹；tfilter 为噪声滤波器的时间常数和未测扰动的滞后时间常数，默认值对应无滤波器和阶跃未测扰动的情形；dplant 为输入不可测扰动模型的阶跃响应系数矩阵；dmodel 为输入可测扰动模型的阶跃响应系数矩阵；对于输入不可测的扰动，dstep 为扰动模型的输出值；对于可测扰动， dstep 为扰动模型的输入；y 为系统的输出；u 为控制变量；ym 为模型预测输出。

对应上述参数的二次型性能指标为

$$J=[\boldsymbol{Y}(k+1)-\boldsymbol{R}(k+1)]^{\mathrm{T}}\boldsymbol{Q}[\boldsymbol{Y}(k+1)-\boldsymbol{R}(k+1)]+\boldsymbol{U}^{\mathrm{T}}(k)\boldsymbol{R}\boldsymbol{U}(k)$$

式中，$\boldsymbol{Y}(k+1)=[y(k+1),y(k+2),\cdots,y(k+p)]^{\mathrm{T}}$

$$\boldsymbol{U}(k)=[u(k),u(k+1),\cdots,u(k+m-1)]^{\mathrm{T}}$$

式中，$\boldsymbol{Q}$ 为加权矩阵 ywt；$\boldsymbol{R}$ 为加权矩阵 uwt。

例 8-12 考虑如下的双输入双输出纯时延对象，其传递函数矩阵为

$$\boldsymbol{G}(s)=\begin{bmatrix}\dfrac{12.8e^{-1s}}{16.7s+1} & \dfrac{6.6e^{-7s}}{10.9s+1}\\ \dfrac{-18.9e^{-3s}}{21.0s+1} & \dfrac{-19.4e^{-3s}}{14.4s+1}\end{bmatrix}$$

解：MATLAB 程序如下：

```
%将传递函数模型转换为阶跃响应模型
g11=poly2tfd(12.8,[16.7 1],0,1);
g12=poly2tfd(6.6,[10.9 1],0,7);
g21=poly2tfd(-18.9,[21.0 1],0,3);
g22=poly2tfd(-19.4,[14.4 1],0,3);
delt=3;
ny=2;tfinal=90;
model=tfd2step(tfinal,delt,ny,g11,g12,g21,g22);
%进行模型预测控制器设计
plant=model;
%预测时域长度为 6
p=6;m=2;
ywt=[];uwt=[1 1];
%设置输入约束和参考轨迹等控制器参数
r=[1 1];
tend=30;                    %仿真时间为 30
ulim=[-0.1 -0.1 0.5 0.5 0.1 100];
```

```
ylim=[];
[y,u,ym]=cmpc(plant,model,ywt,uwt,m,p,tend,r,ulim,ylim);
plotall(y,u,delt)
```

闭环系统的输出和控制量变化曲线如图 8-9 所示。

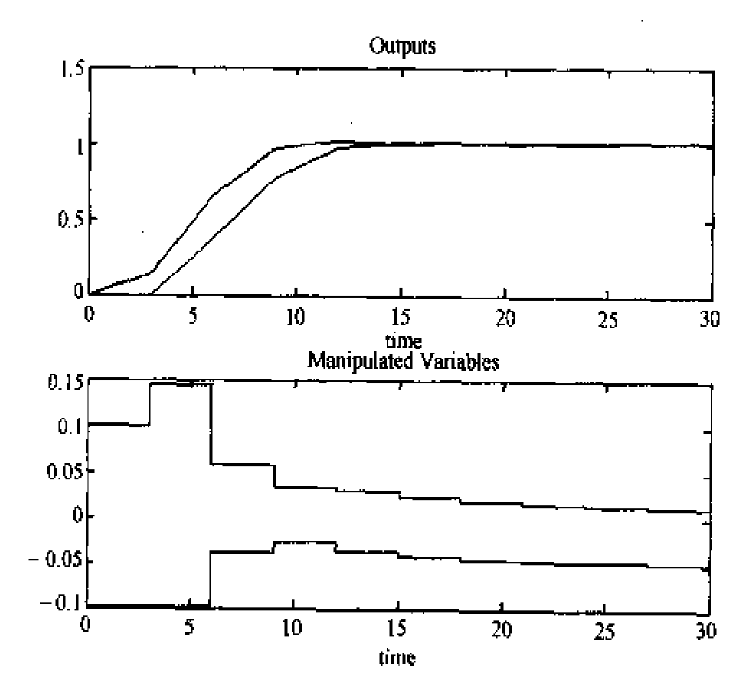


图 8-9　闭环系统输出和控制量变化曲线

8.3.2　输入/输出无约束的模型预测控制器设计

前面介绍了系统输入/输出有约束条件下的模型预测控制器设计问题。对于输入/输出无约束的情况，利用函数 mpccon( )可以完成基于系统阶跃响应模型的预测控制器的设计。下面对该函数进行介绍。

1．输入/输出无约束的模型预测控制器设计函数 mpccon( )

该函数的调用格式为

```
Kmpc=mpccon(model,ywt,uwt,M,P)
```

式中，model 为开环对象的阶跃响应模型；ywt 为二次型性能指标的输出误差加权矩阵；uwt 为二次型性能指标的控制量加权矩阵；M 为控制时域长度；P 为预测时域长度，当 P=Inf 时，表示无限的预测和控制时域长度；Kmpc 为模型预测控制器的增益矩阵。

例 8-13　考虑具有如下传递函数矩阵的系统

$$G(s)=\begin{bmatrix}\dfrac{12.8e^{-1s}}{11.2s+1} & \dfrac{6.6e^{-2s}}{5.2s+1}\\ \dfrac{-1.2e^{-4s}}{3.0s+1} & \dfrac{1.3e^{-5s}}{5s+1}\end{bmatrix}$$

解：下面的MATLAB语句对该系统的阶跃响应模型进行输入/输出无约束的模型预测控制器设计。

```
g11=poly2tfd(12.8,[11.2 1],0,1);g12=poly2tfd(6.6,[5.2 1],0,2);
g22=poly2tfd(1.3,[5 1],0,5);g21=poly2tfd(-1.2,[3.0 1],0,4);
delt=3;ny=2;tfinal=90;
model=tfd2step(tfinal,delt,ny,g11,g12,g21,g22);plant=model;
p=6; m=2;                                    %预测时域长度为6，控制时域长度为2
ywt=[];uwt=[1 1];kmpc=mpccon(model,ywt,uwt,m,p);
r=[0 1];tend=40;
[y,u,ym]=mpcsim(model,model,kmpc,tend,r);   %进行模型预测控制仿真
plotall(y,u,delt)                            %绘制输入/输出变量的变化曲线
```

闭环系统的输出和控制量变化曲线如图8-10所示。

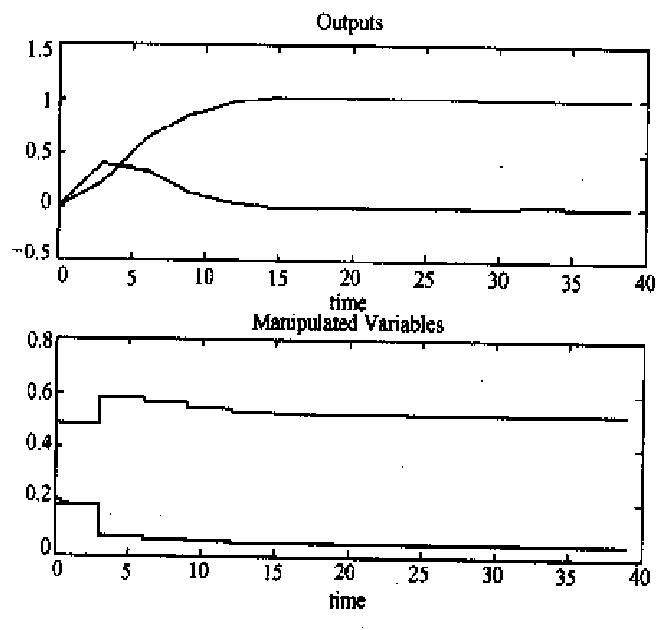


图8-10 闭环系统输出和控制量变化曲线

在上例中，使用了函数 mpccon( )来进行闭环系统的仿真。该函数的功能就是对无约束的模型预测控制系统进行仿真。

2. 输入/输出无约束的模型预测控制系统仿真函数 mpcsim( )

该函数的调用格式为

```
[y,u,ym]=mpcsim(plant,model,kmpc,tend,r,usat, tfilter,dplant,dmodel,dstep)
```

式中，plant 为开环对象的实际阶跃响应模型；model 为辨识得到的开环对象阶跃响应模型；kmpc 为模型预测控制器的增益矩阵；tend 为仿真的结束时间；r 为输入设定值或参考轨迹。以下的输入参数为可选参数：tfilter 为噪声滤波器的时间常数和未测扰动的滞后时间常数，默认值对应无滤波器和阶跃未测扰动的情形；dplant 为输入不可测扰动模型的阶跃响应

系数矩阵；dmodel 为输入可测扰动模型的阶跃响应系数矩阵；对于输入不可测的扰动，dstep 为扰动模型的输出值；对于可测扰动，dstep 为扰动模型的输入；y 为系统的输出；u 为控制变量；ym 为模型预测输出。

8.3.3 计算由阶跃响应模型构成的闭环系统模型

当对象和控制器的模型均由阶跃响应形式给定时，函数 mpccl( )用于计算闭环系统的 MPC 状态空间模型。其使用方法说明如下。

该函数的调用格式为

```
[clmod,cmod]=mpccl(plant,model,kmpc,tfilter,dplant,dmodel)
```

式中，plant 为开环对象的实际阶跃响应模型；model 为阶跃响应形式的内部模型；kmpc 为模型预测控制器的增益矩阵；tfilter,dplant,dmodel 为可选参数，tfilter 为滤波器的时间常数和噪声动力学参数构成的矩阵；dplant 为所有扰动（包括可测扰动和不可测扰动）的阶跃响应模型，若 dplant 为空矩阵，则表示无扰动；dmodel 为可测扰动的阶跃响应模型，若 dmodel 为空矩阵，则表示无可测扰动；clmod 为模型预测控制闭环系统的 MPC 状态空间模型；cmod 为控制器的 MPC 状态空间模型。

例 8-14 考虑如下的 SISO 系统，其传递函数为

$$G(s)=\frac{3e^{-4s}}{4s+1}$$

解：首先利用以下程序绘制开环系统的阶跃响应曲线，如图 8-11 所示。

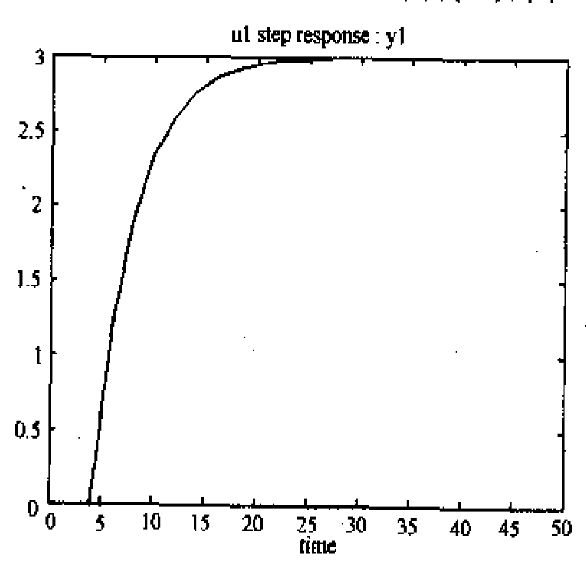


图 8-11 开环系统阶跃响应曲线

```
g=poly2tfd(3,[4 1],0,4);
plant=tfd2step(50,2,1,g);
plotstep(plant)
%设计模型预测控制器
```

```
P=6; %预测时域长度为 6
M=2; %控制时域长度为 2
%设置性能指标的加权阵
ywt=[ ]; uwt=1;
kmpc=mpccon(plant,ywt,uwt,M,P);
[clmod,cmod]=mpccl(plant,plant,kmpc);      %计算闭环系统模型
tend=50; r=3;
[y,u,ym]=mpcsim(plant,plant,kmpc,tend,r);
plotall(y,u,2)
```

闭环系统的阶跃响应曲线和控制量变化曲线如图 8-12 所示。

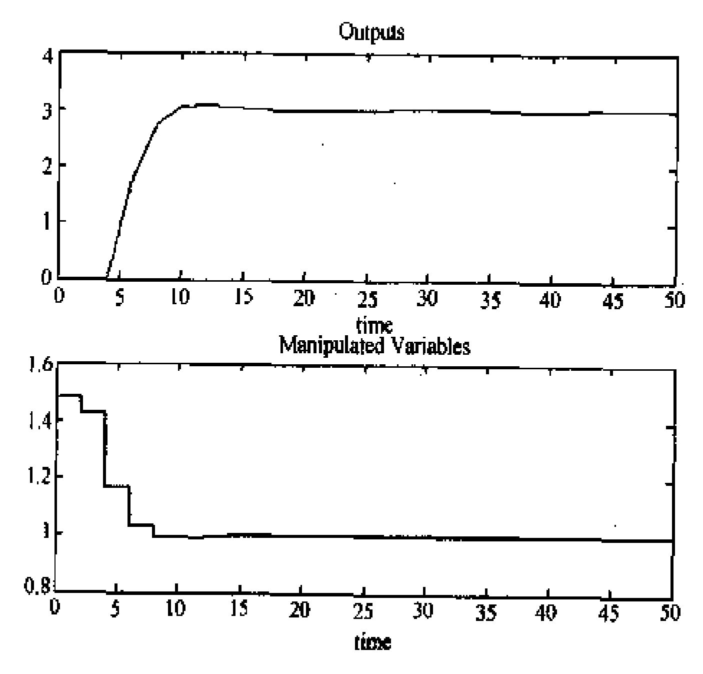


图 8-12 闭环系统阶跃响应曲线和控制量变化曲线

8.4 基于状态空间模型的预测控制器设计函数

在 MATLAB 模型预测控制工具箱中，除了提供基于阶跃响应模型的预测控制器设计功能外，还提供了基于 MPC 状态空间模型的预测控制器设计功能。有关的函数见表 8-4。

表 8-4 基于 MPC 状态空间模型的预测控制器设计函数

函数名	功能
scmpc( )	输入/输出有约束的状态空间模型预测控制器设计
smpccon( )	输入/输出无约束的状态空间模型预测控制器设计
smpccl( )	计算输入/输出无约束的模型预测闭环控制系统模型
smpcsim( )	输入有约束的模型预测闭环控制系统仿真
smpcest( )	状态估计器设计

8.4.1　输入/输出有约束的状态空间模型预测控制器设计

函数 scmpc( )用于进行输入/输出有约束条件下的状态空间模型预测控制器设计，该函数的调用格式为

[y,u,ym]=scmpc(pmod,imod,ywt,uwt,M,P,tend,r,ulim,ylim,kest,z,v,w,wu)

式中，pmod 为 MPC 状态空间模型格式的对象状态空间模型，用于仿真；imod 为 MPC 状态空间模型格式的对象内部模型，用于预测控制器设计；ywt 为二次型性能指标的输出误差加权矩阵；uwt 为二次型性能指标的控制量加权矩阵；M 为控制时域长度；P 为预测时域长度；tend 为仿真的结束时间；r 为输入设定值或参考轨迹；ulim=[ulow uhigh delu]，式中，ulow 为控制变量的下界，uhigh 为控制变量的上界，delu 为控制变量的变化率约束；ylim=［ylow yhigh］，其中 ylow 为输出的下界，yhigh 为输出的上界；kest 为估计器的增益矩阵；z 为测量噪声，v 为测量扰动；w 为输出未测量扰动；wu 为施加到控制输入的未测量扰动；y 为系统响应；u 为控制变量；ym 为模型预测输出。

例 8-15　设系统的传递函数矩阵为

$$G(s)=\begin{bmatrix}\dfrac{12.8e^{-s}}{16.7s+1} & \dfrac{6.6e^{-7s}}{10.9s+1}\\ \dfrac{-18.9e^{-3s}}{21s+1} & \dfrac{-19.4e^{-3s}}{14.4s+1}\end{bmatrix}$$

解：MATLAB 程序如下：

```
g11=poly2tfd(12.8,[16.7 1],0,1);g21=poly2tfd(6.6,[10.9 1],0,7);
g12=poly2tfd(-18.9,[21.0 1],0,3);g22=poly2tfd(-19.4,[14.4 1],0,3);
delt=3;ny=2;
imod=tfd2mod(delt,ny,g11,g12,g21,g22);
pmod=imod;
p=6;m=2;
ywt=[];uwt=[1 1];
tend=30;
r=[0 1];
ulim=[-inf -0.15 inf inf 0.1 100];
ylim=[];
[y,u]=scmpc(pmod,imod,ywt,uwt,m,p,tend,r,ulim,ylim);
plotall(y,u,delt)
```

闭环系统的输出和控制量变化曲线如图 8-13 所示。

在添加对输出变量的约束后，再利用以下程序对系统进行模型预测控制器的设计，得到的闭环控制系统输出响应和控制量变化曲线，如图 8-14 所示。

```
ulim=[-inf  -0.15  inf  inf  0.1  100];
ylim=[0  0  inf  inf];
[y,u]=scmpc(pmod,imod,ywt,uwt,m,p,tend,r,ulim,ylim);
plotall(y,u,delt)
```

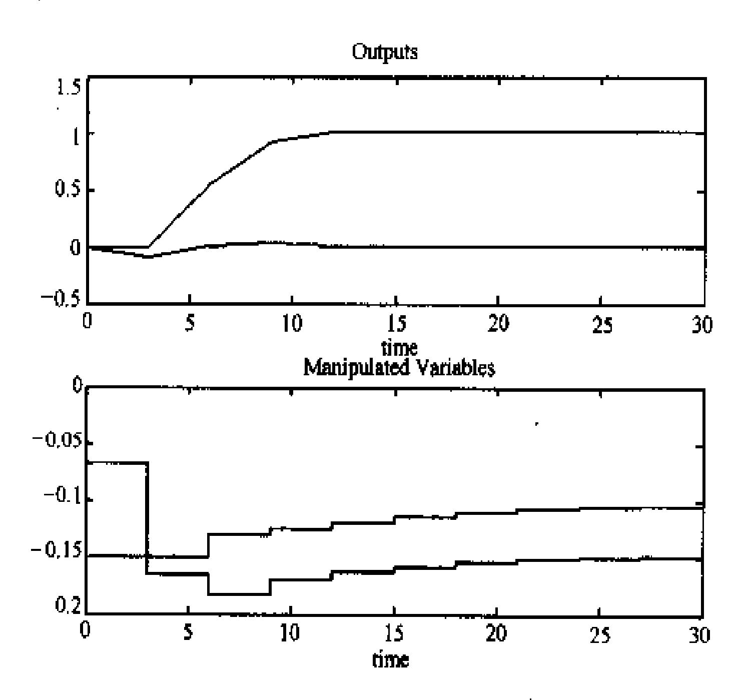


图 8-13 闭环系统输出和控制量变化曲线

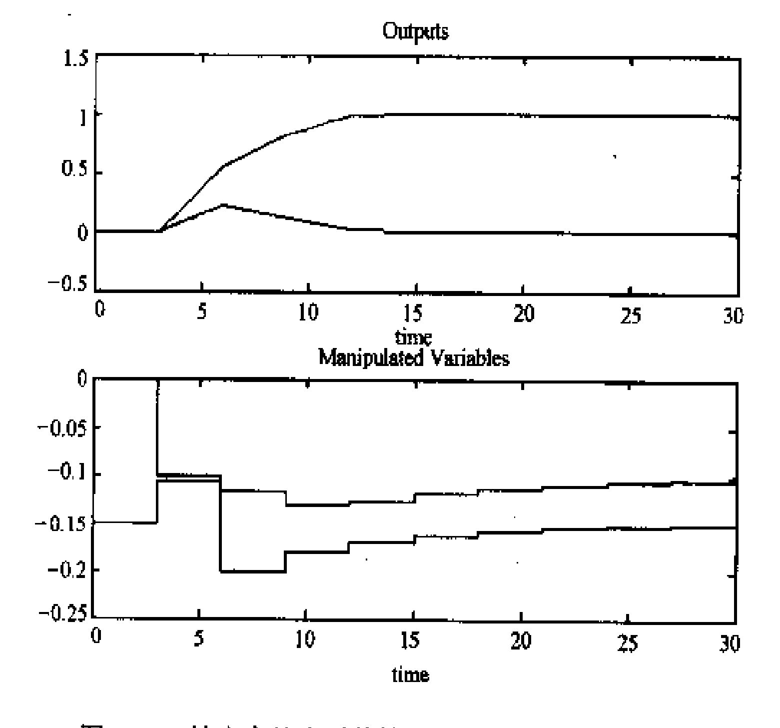


图 8-14 输出有约束时的输出响应和控制量变化曲线

8.4.2　输入/输出无约束的状态空间模型预测控制器设计

函数 smpccon( )用于完成输入/输出无约束的状态空间模型预测控制器设计，其输出为预测控制器的增益矩阵。在这一基础上，利用函数 smpcsim( )可以对模型预测控制闭环系统进行仿真，在仿真时还可以对控制量施加约束。函数 smpccl( )则用于计算闭环系统的 MPC 状态空间模型。

1．输入/输出无约束的状态空间模型预测控制器设计函数 smpccon( )

该函数的调用格式为

```
Ks=smpccon(imod,ywt,uwt,M,P)
```

式中，imod 为 MPC mod 格式的对象内部模型，用于预测控制器设计；ywt 为二次型性能指标的输出误差加权矩阵；uwt 为二次型性能指标的控制量加权矩阵；M 为控制时域长度；P 为预测时域长度；Ks 为预测控制器的增益矩阵。

2．计算输入/输出无约束的模型预测闭环控制系统模型函数 smpccl( )

该函数的调用格式为

```
[clmod,cmod]=smpccl(pmod,imod,Ks)
[clmod,cmod]=smpccl(pmod,imod,Ks,Kest)
```

式中，pmod 为 MPC 状态空间模型；imod 为 MPC 状态空间模型格式的对象内部模型；Ks 为预测控制器的增益矩阵；Kest 为状态估计器的增益矩阵；clmod 为闭环系统的 MPC 状态空间模型；cmod 为预测控制器的 MPC 状态空间模型。

3．输入受限的模型预测控制闭环系统设计与仿真函数 smpcsim( )

该函数的调用格式为

```
[y,u,ym]=smpcsim(pmod,imod,Ks,tend,r,ulim,Kest,z,v,w,wu)
```

式中，pmod 为 MPC 状态空间模型，用于仿真；imod 为 MPC 状态空间模型格式的对象内部模型，用于预测控制器设计；Ks 为预测控制器的增益矩阵；tend 为仿真时间长度；r 为输入设定值；ulim 为输入控制量约束，当 ulim 为 0 或空矩阵时，无输入约束；当对输入施加约束时，ulim=[ulow uhigh delu]; Kest 为估计器增益矩阵；z 为测量噪声；v 为可测扰动（或前馈控制）；w 为输出不可测扰动；wu 为输入不可测扰动；y 为系统响应；u 为控制变量；ym 为模型预测输出。

例 8-16　考虑具有如下传递函数矩阵的多变量系统的状态空间模型预测控制器设计问题。

$$G(s)=\begin{bmatrix}\dfrac{12.8e^{-s}}{16.7s+1} & \dfrac{6.6e^{-7s}}{10.9s+1}\\ \dfrac{-18.9e^{-3s}}{21s+1} & \dfrac{-19.4e^{-3s}}{14.4s+1}\end{bmatrix}$$

解：MATLAB 程序如下：

```
%在进行模型预测控制器设计之前，首先将系统模型转换为状态空间形式
  T=2;
  g11=poly2tfd(12.8,[16.7   1],0,1);
  g12= poly2tfd(6.6,[10.9   1],0,7);
  g21=poly2tfd(-18.9,[21.0   1],0,3);
  g22= poly2tfd(-19.4,[14.4   1],0,3);
  umod=tfd2mod(T,2,g11,g12,g21,g22);
%定义扰动模型
  g13=poly2tfd(3.8,[14.9 1],0,8);
  g23=poly2tfd(4.9,[13.2 1],0,3);
  dmod=tfd2mod(T,2,g13,g23);
%建立叠加了扰动的混合系统模型
  pmod=addumd(umod,dmod);
%考虑精确建模的情况
  imod=pmod;
  ywt=[ ];
  uwt=[ ];
%预测时域和控制时域均为 5
  P=5; M=P;
  Ks=smpccon(imod,ywt,uwt,M,P);
  tend=30;
  r=[1 0];
  [y,u]=smpcsim(pmod,imod,Ks,tend,r);
  plotall(y,u,T)
```

在精确建模的情况下，利用以上程序可得闭环系统的输出和控制量变化曲线如图 8-15 所示。

增加预测时域长度，同时减少控制时域长度后，再利用以下程序可得闭环系统的输出曲线和控制量变化曲线如图 8-16 所示。

```
P=10;                                  %预测时域长度为 10
M=3;                                   %控制时域长度为 3
Ks=smpccon(imod,ywt,uwt,M,P);
[y,u]=smpcsim(pmod,imod,Ks,tend,r);
plotall(y,u,T)                         %绘制输入/输出曲线
```

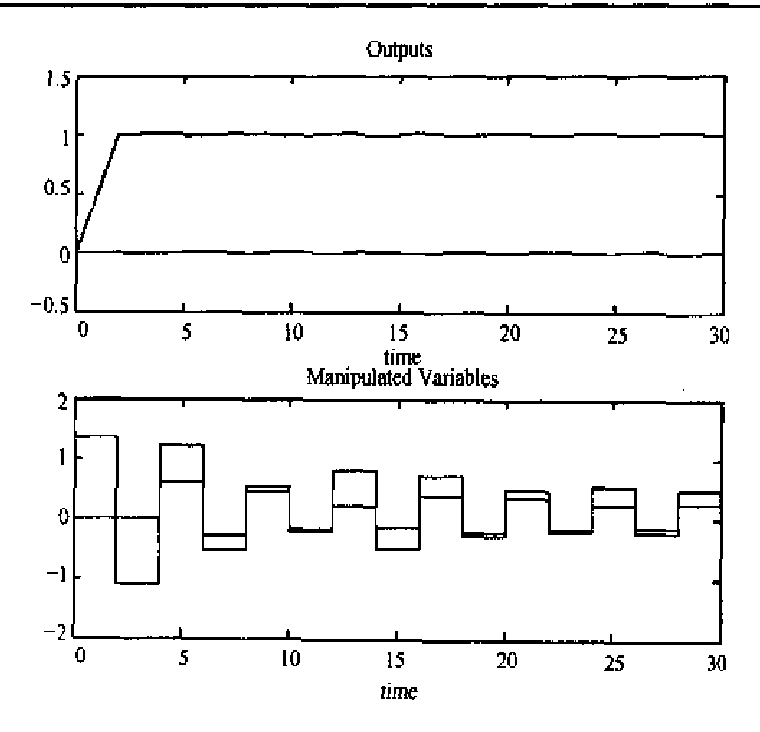


图 8-15　闭环系统输出和控制量变化曲线

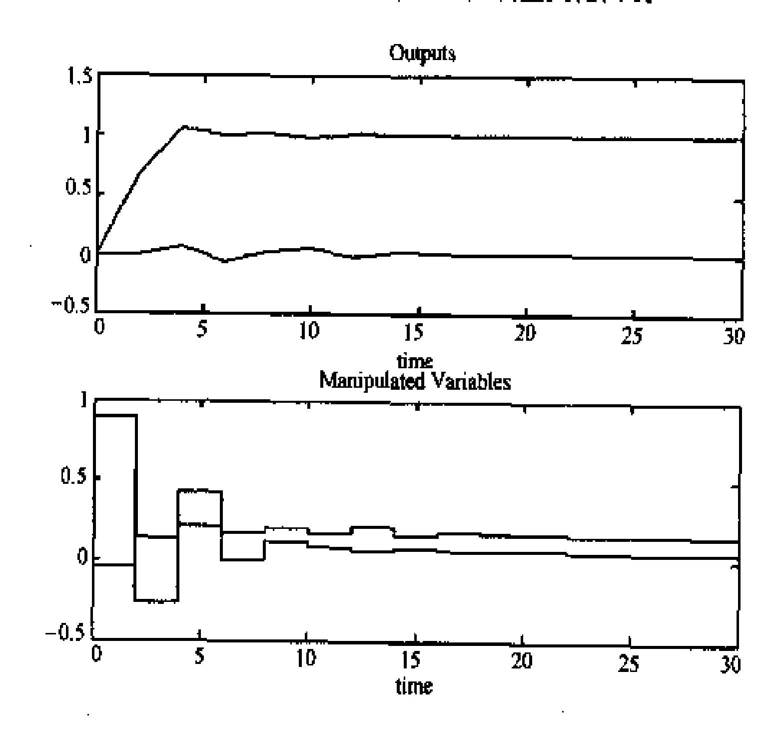


图 8-16　改变预测时域长度后的闭环系统输出和控制量变化曲线

进一步改变控制时域长度，采用控制量分块的形式，即再利用以下程序可得到的闭环系统的输出和控制量变化曲线如图 8-17 所示。

```
M=[2  3  4];
Ks=smpccon(imod,ywt,uwt,M,P);
[y,u]=smpcsim(pmod,imod,Ks,tend,r);
```

```
plotall(y,u,T)
```

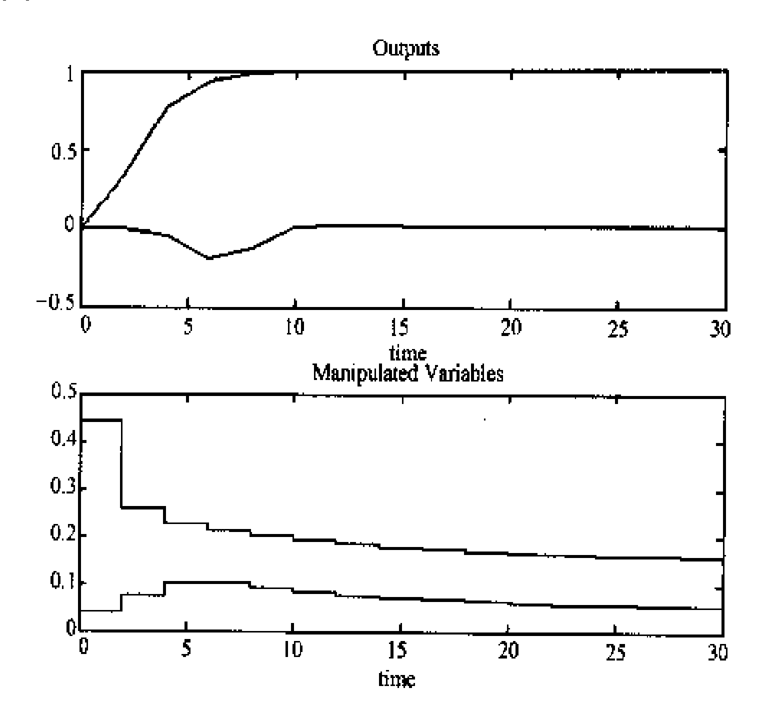


图 8-17 改变控制时域长度后的闭环系统输出和控制量变化曲线

增加输入控制量的加权矩阵系数，即再利用以下程序可得模型预测闭环控制系统输出和控制量变化曲线如图 8-18 所示。

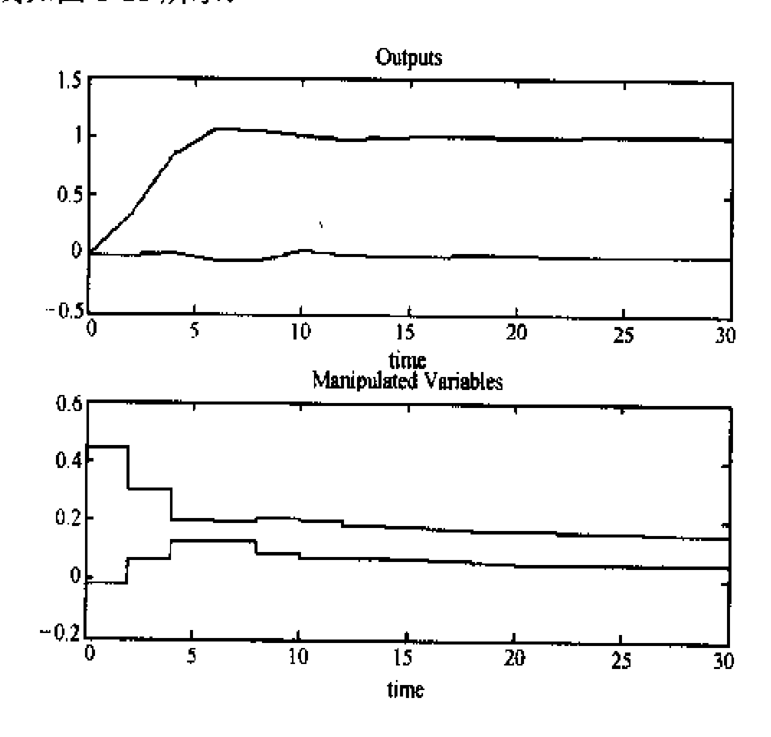


图 8-18 增加控制量加权矩阵后的闭环系统输出和控制量变化曲线

```
uwt=[1    1];
P=5;                                    %预测时域长度为 5
M=P;                                    %控制时域长度等于预测时域长度
Ks=smpccon(imod,ywt,uwt,M,P);
[y,u]=smpcsim(pmod,imod,Ks,tend,r);
plotall(y,u,T)
```

利用以下程序将输出设定值均设为 0，绘制闭环系统的输出和控制量变化曲线，如图 8-19 所示。

```
ulim=[ ];Kest=[ ];R=[ ];
%测量噪声和扰动为 0
z=[ ];v=[ ];w=[ 1];
[y,u]=smpcsim(pmod,imod,Ks,tend,r,ulim,Kest,z,v,w);
plotall(y,u,T)
```

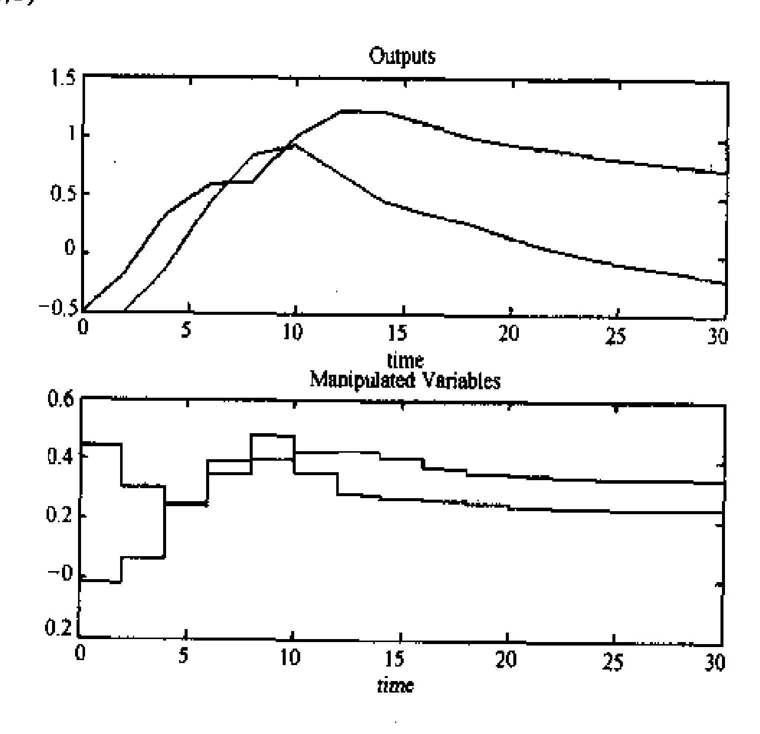


图 8-19 零设定值时的闭环系统输出和控制量变化曲线

利用以下程序采用估计器进一步改善系统性能（参见函数 smpcest( )），对应的系统输出和控制量变化曲线如图 8-20 所示。

```
[Kest,newmod]=smpcest(imod,[15,15],[3 3]);
Ks=smpccon(newmod,ywt,uwt,M,P);
[y,u]=smpcsim(pmod,newmod,Ks,tend,r,ulim,Kest,z,v,w);
plotall(y,u,T)
```

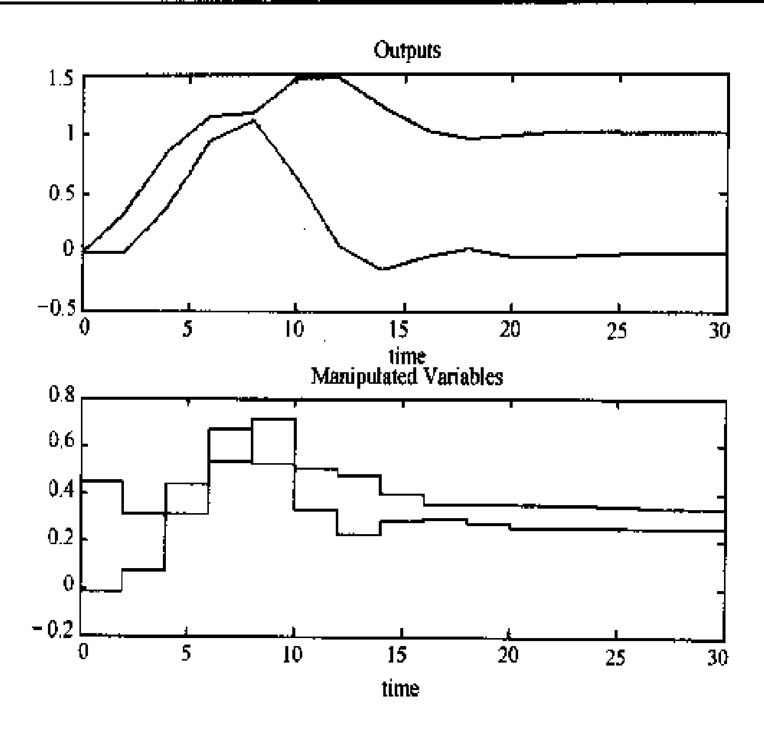


图 8-20 增加估计器后的闭环系统输出和控制量变化曲线

8.4.3 状态估计器设计

函数 smpcest( )用于系统的状态估计器设计，该函数的使用方法说明如下。该函数的调用格式为

[Kest,newmod]=smpcest(imod,tau,signoise)

Kest=smpcest(imod,Q,R)

式中，imod 为系统的内部模型；tau 为时间常数向量，tau 的各个分量用于指定扰动对每个输出的影响特性；signoise 为每个输出的信噪比；Q 为未测量的方差矩阵；R 为测量噪声的方差矩阵；Kest 为状态估计器的增益矩阵；newmod 为修改了的系统模型，在该模型中引入了新的状态来表示扰动的影响。

例 8-17 仍然考虑如下的多变量系统

$$G(s)=\begin{bmatrix}\dfrac{12.8e^{-s}}{16.7s+1} & \dfrac{6.6e^{-7s}}{10.9s+1}\\ \dfrac{-18.9e^{-3s}}{21s+1} & \dfrac{-19.4e^{-3s}}{14.4s+1}\end{bmatrix}$$

解：MATLAB 程序如下：

```
g11=poly2tfd(12.8,[16.7  1],0,1);
g12=poly2tfd(6.6,[10.9  1],0,7);
g21=poly2tfd(-18.9,[21.0  1],0,3);
g22=poly2tfd(-19.4,[14.4  1],0,3);
```

```
delt=1;
ny=2;
imod=tfd2mod(delt,ny,g11,g12,g21,g22);
gw1=poly2tfd(3.8, [14.9   1],0,8);
gw2=poly2tfd(4.9, [13.2   1],0,3);
pmod=addumd(imod,tfd2mod(delt,ny,gw1,gw2));
%设计模型预测控制器
P=6; M=2;
ywt=[ ];uwt=[1   1];
Ks=smpccon(imod,ywt,uwt,M,P);
r=[ ];
ulim=[ ];
z=[ ];
v=[ ];
w=[1];
wu=[ ];
tend=30;
[y3,u3]=smpcsim(pmod,imod,Ks,tend,r,ulim,[],z,v,w,wu);
%设计状态估计器
Kest1=smpcest(pmod,1,0.001*eye(ny));
Ks1=smpccon(pmod,ywt,uwt,M,P);
[y1,u1]=smpcsim(pmod,pmod,Ks1,tend,r,ulim,Kest1,z,v,w,wu);
plotall(y1,u1,delt)
```

利用以上程序可得采用了状态估计器的闭环系统输出和控制量变化曲线如图 8-21 所示。

下面进行简化的状态估计器设计

```
tau=[10   10];
signoise=[3   3];
[Kest2,newmod]=smpcest(imod,tau,signoise);
Ks2=smpccon(newmod,ywt,uwt,M,P);
[y2,u2]=smpcsim(pmod,newmod,Ks2,tend,r,ulim,Kest2,z,v,w,wu);
plotall(y2,u2,delt)
```

再利用以上程序可得采用简化的状态估计器的闭环系统输出和控制量曲线如图 8-22 所示。

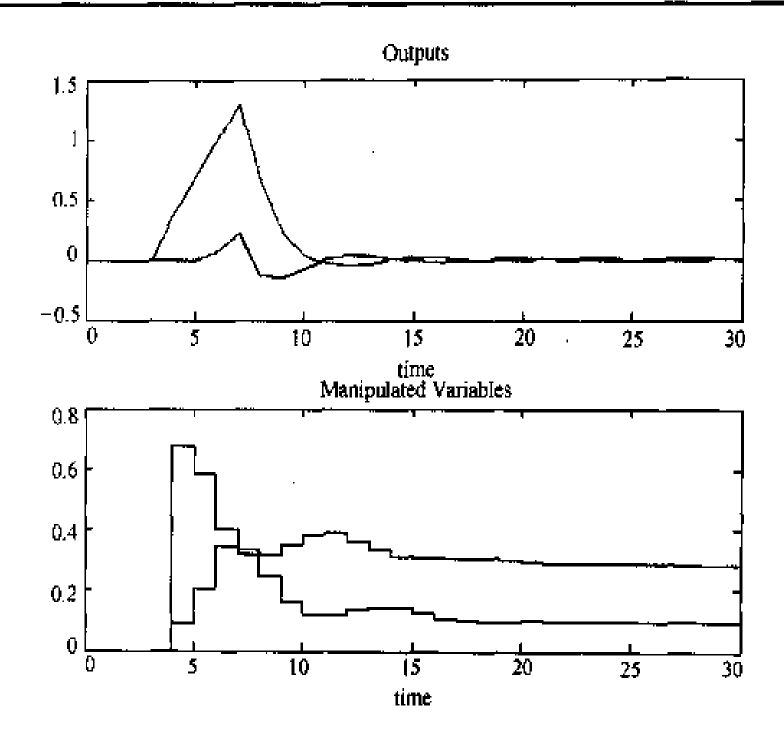


图 8-21　系统输出和控制量变化曲线

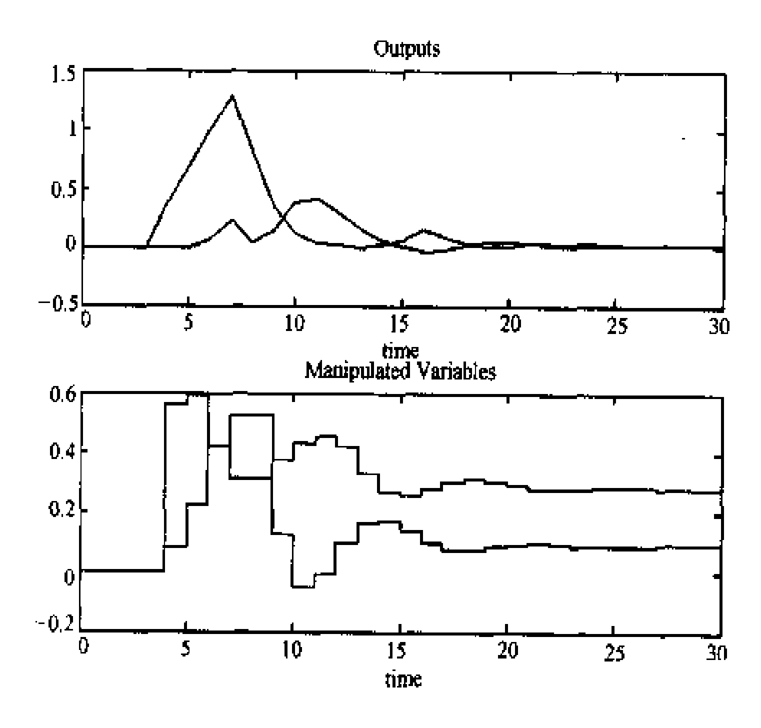


图 8-22　简化状态估计器设计后的系统输出和控制量曲线

8.5 系统分析与绘图函数

前面介绍了模型预测控制工具箱的系统设计与仿真功能函数，这些函数提供了进行模型预测控制器设计的有力工具。为进一步完善其功能，模型预测控制工具箱还包括若干系统

分析和绘图功能函数，见表 8-5。

表 8-5　系统分析与绘图函数

函　数　名	功　　能
mod2frsp( )	计算系统（MPC 状态空间模型）的频率响应
plotfrsp( )	绘制系统的频率响应波特图
svdfrsp( )	计算频率响应的奇异值
smpcpole( )	计算系统（MPC 状态空间模型）的极点
smpcgain( )	计算系统（MPC 状态空间模型）的稳态增益矩阵
mpcinfo( )	输出系统表示的矩阵类型和属性
plotall( )	绘制系统仿真的输入/输出曲线（一个图形窗口）
ploteach( )	在多个图形窗口分别绘制系统的输入/输出仿真曲线
plotstep( )	绘制系统阶跃响应模型的曲线

8.5.1　计算和绘制系统的频率响应曲线

函数 mod2frsp( )和 plotfrsp( )分别用于计算和绘制系统的频率响应特性，其使用说明如下。

1. 计算 MPC 状态空间模型系统的频率响应函数 mod2frsp( )

该函数的调用格式为

```
[frsp,eyefrsp]=mod2frsp(mod,freq,out,in,balflag)
```

式中，mod 为系统的 MPC 状态空间模型；freq 为频率向量，指定对数频率的区间范围和频率点个数；out 为指定输出变量，若 out=[]，则计算所有输出；in 为指定输入变量，若 in=[]，则计算所有输入；balflag 若为非零值，则在计算之前进行系统矩阵的均衡处理；frsp 为系统的输出频率响应矩阵；当频率响应矩阵为方阵时，eyefrsp=I-frsp，其中，I 为单位对角矩阵。

2. 绘制系统的频率响应波特图函数 plotfrsp( )

该函数的调用格式为

```
plotfrsp(vmat)
plotfrsp(vmat,out,in)
```

式中，vmat 为系统的频率响应矩阵；out 为指定输出变量；in 为指定输入变量。

例 8-18　计算并绘制系统的频率响应曲线。设系统的传递函数为

$$G(s)=\frac{3e^{-4s}}{5s+1}$$

解：利用以下 MATLAB 程序，可得如图 8-23 所示频率响应曲线：

```
g=poly2tfd(3,[5 1],0,4);
```

```
mod=tfd2mod(0.1,1,g);
vmat=mod2frsp(mod,[-2,3,50],1,1,0);
plotfrsp(vmat)
```

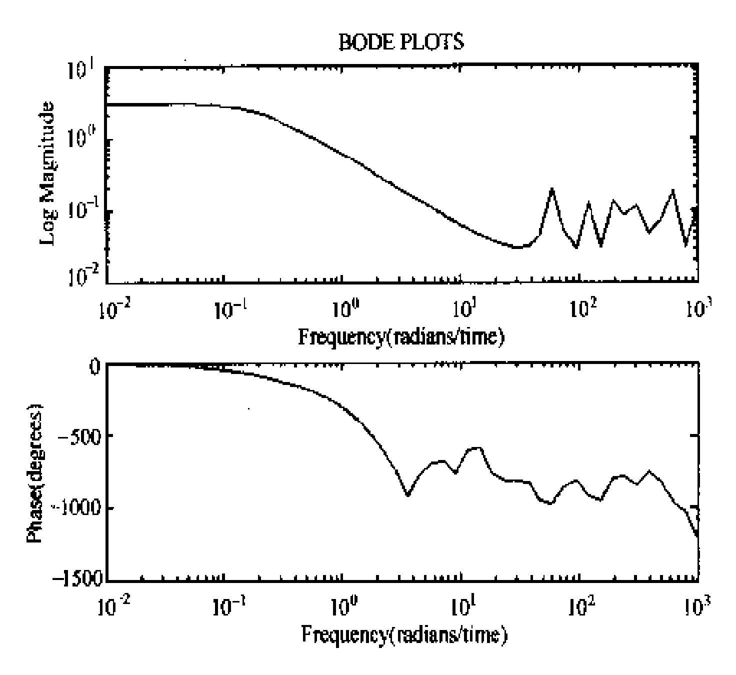


图 8-23 频率响应 Bode 曲线

8.5.2 计算频率响应的奇异值

在获得系统的频率响应矩阵后，函数 svdfrsp( )用于计算系统频率响应矩阵的奇异值，其使用方法如下。该函数的调用格式为

[sigma,omega]=svdfrsp(vmat)

式中，vmat 为系统的频率响应矩阵；sigma 为频率响应的奇异值矩阵，sigma 的第 i 行为第 i 个频率响应子矩阵的按降序排列的奇异值；omega 为包含频率响应矩阵的独立变量值的列向量。

8.5.3 计算系统的极点和稳态增益矩阵

1. 计算系统的极点函数 smpcpole( )

该函数的调用格式为

poles=smpcpole(mod)

式中，mod 为系统的 MPC 状态空间模型；poles 为以复数向量给出的系统的极点。

2. 计算系统的稳态增益矩阵函数 smpcgain( )

该函数的调用格式为

g=smpcgain(mod)

式中，mod 为开环稳定的对象模型（MPC mod 格式）；g 为系统的稳态增益矩阵，该矩阵的行对应输出变量，列对应输入变量。

例
```
>>g=poly2tfd(2,[4  1],0,3);
>>mod=tfd2mod(0.1,1,g);
>>[poles]=smpcgain(mod)
```
其结果为
```
poles =
        2.0000
```

8.5.4 系统分析和绘图

1. 返回系统模型矩阵的信息函数 mpcinfo( )

该函数的调用格式为

flag=mpcinfo(mat)

式中，mat 为系统模型矩阵；flag 为矩阵的类型，flag<0 为常数矩阵；flag=1 为系统矩阵；flag=2 为时变矩阵；flag=4 为 MPC 状态空间模型；flag=5 为 MPC 阶跃响应模型。

例
```
>>g=poly2tfd(2,[4  1],0, 3 );
>>mod=tfd2mod(0.1,1,g);
>>flag1=mpcinfo(g);
>>flag2=mpcinfo(mod)
```
其输出结果为
```
flag1=
     -3
flag2=
     4
```

2. 绘制系统仿真的输入/输出曲线函数 plotall( )

该函数的调用格式为

plotall(y,u)

plotall(y,u,t)

式中，y 和 u 为系统的输入/输出变量，其中控制量 u 在绘图之前转换为阶梯型连续数据变量；t 为采样周期。

3. 在多个窗口绘制系统仿真的输入/输出曲线函数 ploteach( )

该函数的调用格式为

ploteach(y)

ploteach(y,u,t)

ploteach(y,u)

ploteach([ ],u)

ploteach(y,[ ],t)

输入参数定义与函数 plotall( )相同。

例 8-19 绘制系统的阶跃响应曲线，其中系统的传递函数为

$$G(s)=\frac{2e^{-3s}}{4s+1}。$$

解：利用以下 MATLAB 程序可得如图 8-24 所示系统的响应曲线。

```
g=poly2tfd(2, [4, 1], 0, 3);
mod=tfd2mod(2, 1, g);
P=7; M=4;
ywt=[ ];
uwt=1;
tend=30;
r=2;
ulim=[-inf   inf   100];
ylim=[ ];delt=2;
[y,u]=scmpc(mod,mod,ywt,uwt,M,P,tend,r,ulim,ylim);
ploteach(y,u,delt)
```

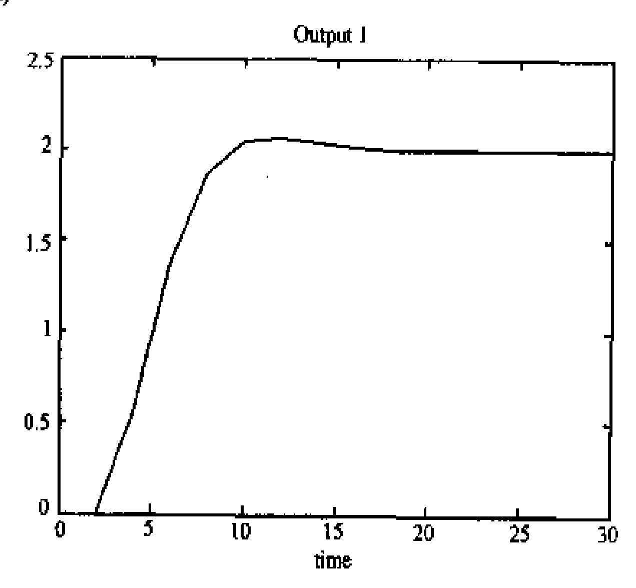


图 8-24 系统阶跃响应曲线

4．绘制系统阶跃响应模型的曲线函数 plotstep( )

该函数的调用格式为

plotstep(plant)

plotstep(plant,opt)

式中，plant 为对象的阶跃响应模型；opt 为指定输出变量。

例 8-20　绘制系统的阶跃响应曲线，其中系统的传递函数为

$$G(s)=\frac{2e^{-3s}}{4s+1}。$$

解：利用以下 MATLAB 程序可得如图 8-25 所示系统的阶跃响应曲线。

```
g=poly2tfd(2,[4  1],0,3); plant=tfd2step(30,2,1,g);
plotstep(plant)
```

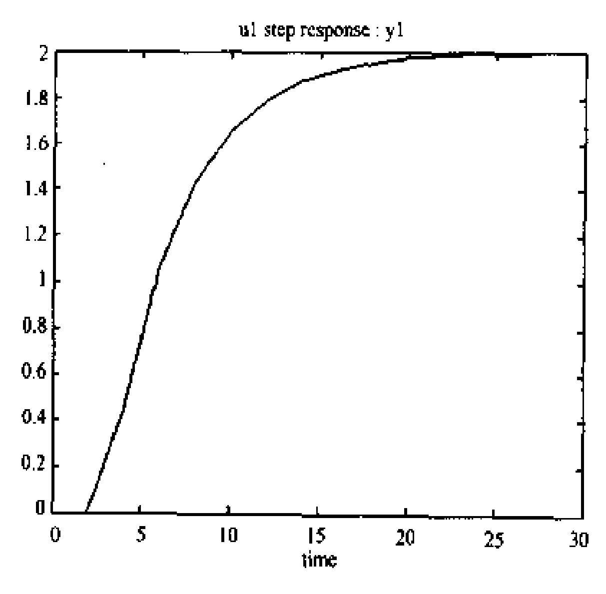


图 8-25　系统阶跃响应曲线

8.6　通用功能函数

模型预测控制工具箱的通用功能函数包括模型转换函数、离散系统分析及仿真函数、分解函数等，见表 8-6。

表 8-6　模型预测控制工具箱通用功能函数

函 数 名	功　能
abcdchkm( )	检查状态空间矩阵 A,B,C,D 的维数一致性
cp2dp( )	将连续型多项式传递函数转换为离散型多项式传递函数
c2dmp( )	连续系统离散化
d2cmp( )	将离散系统转换为连续函数
mpcaugss( )	增广状态空间模型
mpcparal( )	将两个状态空间模型并联
ss2tf2( )	将状态空间模型转换为传递函数
tf2ssm( )	将传递函数转换为状态空间模型
dantzgmp( )	求解二次规划问题
dareiter( )	求解离散 Riccati 方程
dlqe2( )	计算离散系统的状态估计器增益矩阵
dlsimm( )	离散系统仿真
dimpulsm( )	生成离散系统的脉冲响应
mpcstair( )	生成阶梯型控制变量
parpart( )	分割参数用于 Simulink 仿真

8.6.1 通用模型转换

1．将连续型多项式传递函数转换为离散型多项式传递函数的函数 cp2dp( )

该函数的调用格式为

[numd,dend]=cp2dp(num,den,delt,delay)

式中，num 为连续传递函数的分子多项式；den 为连续传递函数的分母多项式；delt 为采样周期；delay 为系统时延；numd 为离散传递函数的分子多项式；dend 为离散传递函数的分母多项式。

2．连续系统离散化函数 c2dmp( )

该函数的调用格式为

[G,H,f0]=c2dmp(A,B,T,dx0)

式中，A 和 B 为连续系统状态空间矩阵；T 为采样周期；dx0 为可选参数，默认值为 0，用于不稳定条件下的模型线性化。G 和 H 为离散系统的状态空间矩阵；f0 为对应 dx0 的离散化参数。

3．离散系统转换为连续系统函数 d2cmp( )

该函数的调用格式为

[A,B]=d2cmp(G,H,T)

式中，G 和 H 为离散系统的状态空间矩阵；T 为采样周期；A 和 B 为连续系统的状态空间矩阵。

4．增广状态空间模型函数 mpcaugss( )

该函数的调用格式为

[Aa,Ba,Ca]=mpcaugss(A,B,C)

[Aa,Ba,Ca,Da]=mpcaugss(A,B,C,D)

式中，A,B,C,D 为系统 MPC 状态空间模型矩阵；Aa,Ba,Ca,Da 为增广系统的 MPC 状态空间矩。

5．并联两个状态空间模型函数 mpcparal( )

该函数的调用格式为

[A,B,C,D]= mpcparal(A1,B1,C1,D1,A2,B2,C2,D2)

式中，A1,B1,C1,D1 为子系统 1 的状态空间矩阵；A2,B2,C2,D2 为子系统 2 的状态空间矩阵；A,B,C,D 为并联系统的状态空间矩阵。

6．状态空间模型转换为传递函数模型函数 ss2tf2( )

该函数的调用格式为

[num,den]=ss2tf2(A,B,C,D,iu)

式中，A,B,C,D 为系统状态空间矩阵；iu 为指定输入变量；num 和 den 为传递函数模型。

7．传递函数模型转换为状态空间模型函数 tf2ssm( )

该函数的调用格式为

[A,B,C,D]=tf2ssm(num,den)

式中，num 为传递函数的分子多项式系数向量；den 为传递函数分母多项式系数向量；A,B,C,D 为系统状态空间模型矩阵。

8.6.2 方程求解

1．求解二次规划问题函数 dantzgmp( )

该函数的调用格式为

[bas,ib,il,iter,tab]=dantzgmp(tabi,basi,ibi,ili)

式中，basi 为初始基向量；bas 为最终的基向量；iter 为迭代次数；il 为 Lagrange 乘子的索引向量。

2．求解离散 Riccati 方程函数 dareiter( )

该函数的调用格式为

X=dareiter(G,H,C,D)

式中，X 为离散 Riccati 方程的解；G,H,C,D 为离散 Riccati 方程的系数矩阵，离散 Riccati 方程具有如下形式，即

$$\boldsymbol{G}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{G}-\boldsymbol{X}-\boldsymbol{G}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{H}(\boldsymbol{C}+\boldsymbol{H}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{H})^{-1}\boldsymbol{H}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{G}+\boldsymbol{D}=0$$

8.6.3 离散系统的分析

1．计算离散系统的状态估计器增益矩阵函数 dlqe2( )

函数 dlqe2 利用迭代方法求解 Kalman 滤波器方程。该函数的调用格式为

K=dlqe2(G,Gamw,C,q,r)

[K,m,P]=dlqe2(G,Gamw,C,q,r)

式中，G 和 C 为对象的 MPC 状态空间矩阵；Gamw 为不可测扰动的特性矩阵；q 为不可测扰动的方差矩阵；r 为测量噪声方差矩阵；K 为离散 Kalman 滤波器的稳态增益矩阵；m 为在测量值更新之前的状态估计期望方差；P 为在测量值更新之后的状态估计期望方差。

2．离散系统仿真函数 dlsimm( )

该函数的调用格式为

[y,x]=dlsimm(G,H,C,D,u,x0)

[y,x]=dlsimm(num,den,u)

式中，G,H,C,D 为离散系统的状态空间模型；num 和 den 为离散系统的传递函数模型；u 为输入变量；x0 为初始条件；y 为离散系统的输出响应；x 为离散系统的状态响应。

第 9 章　隐式广义预测自校正控制及其 MATLAB 实现

广义预测控制作为一种新型的远程预测控制方法，集多种算法的优点为一体，具有较好的性能，受到人们的重视。现有多种修正算法，大体上可分为显式算法和隐式算法两种。显式算法是先辨识对象模型参数，然后利用 Diophantine 方程作中间运算，最后得到控制律参数，由于要作多步预测，就必须多次求解 Diophantine 方程，因要经过繁琐的中间运算，故计算工作量较大，占线时间太长。隐式算法不辨识对象模型参数，而是根据输入/输出数据直接辨识求取最优控制律中的参数，因而避免了在线求解 Diophantine 方程所带来的大量中间运算，减少了计算工作量，节省了时间。

本章利用 GPC 并列预报器间的特点，直接辨识预报器中最远程输出预报式中的参数，并利用 GPC 与 DMC 控制律的等价性，来推求最优控制律的参数，提出了一种简单的隐式自校正算法。它保留了 GPC 鲁棒性强等特点，可适用于任何稳定的最小相位/非最小相位和已知/未知时延系统。

当被控对象的参数未知或系统具有慢时变时，根据前面导出的控制律，可以得到相应的自校正控制器算法。

9.1　单输入单输出系统的隐式广义预测自校正控制算法

1．并列预测器

GPC 的最优化控制律

$$\Delta \boldsymbol{U}=(\boldsymbol{G}^{\mathrm{T}}\boldsymbol{G}+\lambda \boldsymbol{I})^{-1}\boldsymbol{G}^{\mathrm{T}}(\boldsymbol{W}-\boldsymbol{f}) \tag{9-1}$$

要求$\Delta \boldsymbol{U}$ 必须知矩阵 $\boldsymbol{G}$ 和开环预测向量 $\boldsymbol{f}$，原因是控制量加权因子λ 和经柔化后的设定值向量 $\boldsymbol{W}$ 均属已知量。隐式自校正方法就是利用输入/输出数据，根据预测方程直接辨识 $\boldsymbol{G}$ 和 $\boldsymbol{f}$。

根据式（7-25）可得 n 个并列预测器为

$$\begin{cases} y(k+1)=g_0\Delta u(k)+f(k+1)+\boldsymbol{E}_1\xi(k+1) \\ y(k+2)=g_1\Delta u(k)+g_0\Delta u(k+1)+f(k+2)+\boldsymbol{E}_2\xi(k+2) \\ \quad\cdots \\ y(k+n)=g_{n-1}\Delta u(k)+\cdots+g_0\Delta u(k+n-1)+f(k+n)+\boldsymbol{E}_n\xi(k+n) \end{cases} \tag{9-2}$$

分析式（9-2）可知，矩阵 $\boldsymbol{G}$ 中所有元素 $g_0,g_1,\cdots,g_{n-1}$ 都在最后一个方程中出现，因此仅对式（9-2）的最后一个方程辨识，即可求得矩阵 $\boldsymbol{G}$。

2. 矩阵 G 的求取

由式（9-2）最后一个方程得

$$y(k+n)=g_{n-1}\Delta u(k)+\cdots+g_0\Delta u(k+n-1)+f(k+n)+E_n\xi(k+n) \tag{9-3}$$

令

$$\boldsymbol{X}(k)=[\Delta u(k),\Delta u(k+1),\cdots,\Delta u(k+n-1),1]$$
$$\boldsymbol{\theta}(k)=[g_{n-1},g_{n-2},\cdots,g_0,f(k+n)]^{\mathrm{T}}$$

则式（9-3）可写为

$$y(k+n)=\boldsymbol{X}(k)\boldsymbol{\theta}(k)+E_n\xi(k+n) \tag{9-4}$$

输出预测值为

$$y(k+n/k)=\boldsymbol{X}(k)\boldsymbol{\theta}(k)$$

或

$$y(k/k-n)=\boldsymbol{X}(k-n)\boldsymbol{\theta}(k) \tag{9-5}$$

若在时刻 k, $\boldsymbol{X}(k-n)$元素已知，$E_n\xi(k+n)$为白噪声，就能用普通最小二乘法估计参数向量 $\boldsymbol{\theta}(k)$，然而通常 $E_n\xi(k+n)$不是白噪声，因此采用将控制策略与参数估计相结合的方法，即用辅助输出预测的估计值 $\hat{y}(k/k-n)$ 来代替输出预测值 $y(k/k-n)$，且认为 $\hat{y}(k/k-n)$ 与实际值 $y(k)$之差为白噪声$\varepsilon(k)$。

由

$$\hat{y}(k/k-n)+\varepsilon(k)=y(k/k-n)+E_n\xi(k)$$
$$y(k)-\hat{y}(k/k-n)=\varepsilon(k)$$

得

$$y(k)=\hat{\boldsymbol{X}}(k-n)\boldsymbol{\theta}(k)+\varepsilon(k) \tag{9-6}$$

$\boldsymbol{\theta}(k)$可用以下递推最小二乘公式估计为

$$\begin{aligned}
\hat{\boldsymbol{\theta}}(k)&=\hat{\boldsymbol{\theta}}(k-1)\boldsymbol{K}(k)[y(k)-\hat{\boldsymbol{X}}(k-1)\hat{\boldsymbol{\theta}}(k-1)]\\
\boldsymbol{K}(k)&=\boldsymbol{P}(k-1)\hat{\boldsymbol{X}}^{\mathrm{T}}(k-n)[\lambda_1+\hat{\boldsymbol{X}}(k-n)\boldsymbol{P}(k-1)\hat{\boldsymbol{X}}^{\mathrm{T}}(k-n)]^{-1}\\
\boldsymbol{P}(k)&=[\boldsymbol{I}-\boldsymbol{K}(k)\hat{\boldsymbol{X}}(k-n)]\boldsymbol{P}(k-1)/\lambda_1
\end{aligned} \tag{9-7}$$

式中，λ_1为遗忘因子，$0<\lambda_1<1$。

利用上述递推公式所得$\boldsymbol{\theta}(k)$的估计值$\hat{\boldsymbol{\theta}}(k)$，即可得到矩阵 $\boldsymbol{G}$ 中的元素 $g_0,g_1,\cdots,g_{n-1}$ 和 $f(k+n)$。

k 时刻 n 步估计值可由下式算出，即

$$\hat{y}(k+n/k)=\hat{\boldsymbol{X}}(k)\hat{\boldsymbol{\theta}}(k) \tag{9-8}$$

式中，$\hat{\boldsymbol{X}}(k)=[\Delta u(k),\Delta u(k+1),\cdots,\Delta u(k+n-1),1]$。

这里，$\Delta u(k),\Delta u(k+1),\cdots,\Delta u(k+n-1)$ 用上一步计算得到的相应点上的控制增量代替。

3. 预测向量 f 的求取

根据 7.3.2 节中所述的 GPC 与 DMC 控制规律的等价性，GPC 中的 f 向量相等于 DMC 中的 $\boldsymbol{Y}_0$ 向量。

可根据式（7-16）得到下一时刻的 Y_0 向量为

$$\begin{bmatrix} y_0(k+1) \\ y_0(k+2) \\ \vdots \\ y_0(k+p-1) \\ y_0(k+p) \end{bmatrix} = \begin{bmatrix} \hat{y}(k+2/k) \\ \hat{y}(k+3/k) \\ \vdots \\ \hat{y}(k+p/k) \\ \hat{y}(k+p/k) \end{bmatrix} + \begin{bmatrix} h_2 \\ h_3 \\ \vdots \\ h_p \\ h_p \end{bmatrix} e(k+1) \tag{9-9}$$

式中，p 为模型时域长度($p \geqslant n$)；$h_2,h_3,\cdots,h_p$ 为误差校正系数；$e(k+1)=y(k+1)-y(k+1/k)$为预测误差，在这里取 $h_2=h_3=\cdots=h_p=1$。

由f与 Y_0 的等价性，利用式（9-9），可得到下一时刻的预测向量f为

$$f = \begin{bmatrix} f(k+1) \\ f(k+2) \\ \vdots \\ f(k+n) \end{bmatrix} = \begin{bmatrix} \hat{y}(k+2/k) \\ \hat{y}(k+3/k) \\ \vdots \\ \hat{y}(k+n+1/k) \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} e(k+1) \tag{9-10}$$

在 G 和 f 求得后，就可利用式（9-1）计算控制量，在计算的每一步，都能得到此步至以后 n 步各点上的 n 个控制序列。为及时利用反馈信息决定控制量，每次仅将序列中第一个控制量作用于系统，其后的 $n-1$ 个控制量不直接作用于系统，而只用于$\hat{Y}$ 的计算。

4. 控制律的简化

在自校正方案中，可从式（9-1）看出，每次计算必须在线求解一次 $n \times n$ 维逆阵 $(G^TG+\lambda I)^{-1}$，在这里与基本的 GPC 一样，也引入控制时域长度 $m(m \leqslant n)$，当 $j>m$ 时，有 $\Delta u(k+j-i)=0$，从而矩阵 G 变成 $n \times m$ 维，矩阵$(G^TG+\lambda I)$ 则变成了 $m \times m$ 方阵，降低了维数，减少了计算工作量。对阶数较低较易控制的简单系统，可取 $m=1$，这时$(G^TG+\lambda I)$将由矩阵变成一个标量数值，而整个运算过程将不会有矩阵运算。

5. GPC 控制算法中主要参数对系统性能的影响

系统的动态过程主要取决于模型精确度和控制参数的设计，对 GPC 来说，影响其性能的参数主要有以下几个。

1）采样周期 T

采样周期 T 直接影响到 $g_0,g_1,\cdots,g_{n-1}$ 和 g^T 矩阵。采样周期 T 的选择，原则上应使采样频率满足香农定理的要求，即采样频率应大于 2 倍截止频率。T 大有利于控制稳定，但不利于抑制扰动。采样周期太长，将会丢失一些有用的高频信息，无法重构出连续时间信号，且使模型不准，控制质量下降；采样周期也不能太短，否则机器计算不过来，且有可能出现离散非最小相位零点，影响闭环系统的稳定。

2）预测长度 n

预测长度 n 对系统的稳定性有重要的影响。若控制长度 m 很小，控制加权系数$\lambda=0$，即

在控制增量不受压制的情况下，增大 n 总可以得到稳定控制；若 m 为任意，通过加大λ，当 $\sum g_i>0$ 时，可得到稳定控制。

n 对系统的动态特性也有影响。当 n 取值较小时，系统的动态性能较差，增加 n，可明显改善系统的动态性能，增强系统的鲁棒性。一般 n 的选择应使最优化时域 $t_p=nT$ 包含对象的主要动态特性，但 n 过大，对进一步改善系统的动态性能作用不大，反而会增加计算时间，一般取 $n=5\sim15$。

3）控制长度 m

控制长度 m 对系统的性能影响较大。较小的 m，对控制起到一定的约束作用，使输出变化平缓，有利于控制系统稳定；而偏大的 m，表示有较多步的控制增量变化，增大了系统的灵活性和快速性，但往往产生振荡和超调，引起系统的不稳定。因此，m 的选择应兼顾快速性与稳定性，一般取 $m=1\sim3$。由于 m 的增加，会使计算时间大大增加，故对于阶数较低较易控制的简单系统，通常取 $m=1$。

4）控制加权系数λ

目标函数中第二项的引入，主要用于压制过于剧烈的控制增量，以防止系统超出限制范围或发生剧烈振荡。增加λ，控制量减少，输出响应速度减慢，有益于增强系统的稳定性；但过大的λ会使控制量的变化极为缓慢，系统得不到及时的调节，反而会使动态特性变坏，一般取 $0<\lambda<1$。$\lambda=0$ 时，对控制量无约束。

5）柔化系数α

柔化系数α 对系统的鲁棒性有重要的影响。由式（7-19）知，若α 小，则 $w(k)$很快趋向 y_r，这时，跟踪的快速性好，鲁棒性差；增加α，系统的快速性变差，而鲁棒性提高，故 α 的选择必须在动态品质与鲁棒性之间折中考虑，一般取 $0<\alpha<1$。

由于 GPC 是根据输出预报信息来进行控制决策的，因此输出预报的精度直接影响到它的控制效果。

9.2　多输入多输出系统的隐式广义预测自校正控制算法

前面提出的单输入单输出系统隐式广义预测控制，虽然对于多输入多输出系统可直接进行推广，但由于输入/输出之间的耦合作用，实际上实现比较困难。本节通过将目标函数分散化和把输入/输出的交互影响前馈解耦，使多变量系统的综合设计问题转化为单输入单输出系统的设计问题，也就是说，MIMO 系统各回路间耦合的影响可看做是前馈输入量。

下面的讨论假定输入/输出都是二维的，大于二维的推导过程与此类同。

1. 系统模型

设 MIMO 的 CARIMA 模型为

$$\begin{bmatrix} A_1(z^{-1}) & 0 \\ 0 & A_2(z^{-1}) \end{bmatrix}\begin{bmatrix} y_1(k) \\ y_2(k) \end{bmatrix}=\begin{bmatrix} \boldsymbol{B}_{11}(z^{-1}) & \boldsymbol{B}_{12}(z^{-1}) \\ \boldsymbol{B}_{21}(z^{-1}) & \boldsymbol{B}_{22}(z^{-1}) \end{bmatrix}\begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix}+\begin{bmatrix} \xi_1(k)/\Delta \\ \xi_2(k)/\Delta \end{bmatrix} \tag{9-11}$$

式中，$A_1(z^{-1}), A_2(z^{-1}), B_{11}(z^{-1}), B_{12}(z^{-1}), B_{21}(z^{-1}), B_{22}(z^{-1})$均为 z^{-1} 的多项式；$y_1(k)$和 $y_2(k)$为系统输出；$u_1(k)$和 $u_2(k)$为系统输入；$\xi_1(k)$和$\xi_2(k)$为白噪声；$\Delta=1-z^{-1}$。

式（9-11）可分解为两个子系统

$$A_1(z^{-1})y_1(k)=B_{11}(z^{-1})u_1(k-1)+B_{12}(z^{-1})u_2(k)+\xi_1(k)/\Delta \tag{9-12}$$

$$A_2(z^{-1})y_2(k)=B_{21}(z^{-1})u_1(k-1)+B_{22}(z^{-1})u_2(k)+\xi_2(k)/\Delta \tag{9-13}$$

1）子系统 1

由式（9-12）的子系统 1 模型与丢番图方程

$$1=E_{1j}(z^{-1})A_1(z^{-1})\Delta+z^{-j}F_{1j}(z^{-1}) \tag{9-14}$$

可得预测方程

$$\begin{aligned} y_1(k+j)=&E_{1j}(z^{-1})B_{11}(z^{-1})\Delta u_1(k+j-1)+E_{1j}(z^{-1})B_{12}(z^{-1})\Delta u_2(k+j-1)\\ &+F_{1j}(z^{-1})y_1(k)+E_{1j}(z^{-1})\xi_1(k+j) \end{aligned}$$

$$(j=1,2,\cdots,n) \tag{9-15}$$

最优输出预测为

$$\hat{y}_1(k+j)=G_{11j}(z^{-1})\Delta u_1(k+j-1)+G_{12j}(z^{-1})\Delta u_2(k+j-1)+F_{1j}(z^{-1})y_1(k)$$

$$(j=1,2,\cdots,n) \tag{9-16}$$

其中

$$G_{11j}=E_{1j}B_{11}=g_{11j0}+g_{11j1}z^{-1}+\cdots+g_{11jj}z^{-j+1}+\cdots$$

$$G_{12j}=E_{1j}B_{12}=g_{12j0}+g_{12j1}z^{-1}+\cdots+g_{12jj}z^{-j+1}+\cdots$$

根据式（9-14）得

$$G_{11j}(z^{-1})=E_{1j}(z^{-1})B_{11}(z^{-1})=\frac{B_{11}(z^{-1})}{A_1(z^{-1})\Delta}[1-z^{-j}F_{1j}(z^{-1})]$$

$$G_{12j}(z^{-1})=E_{1j}(z^{-1})B_{12}(z^{-1})=\frac{B_{12}(z^{-1})}{A_1(z^{-1})\Delta}[1-z^{-j}F_{1j}(z^{-1})]$$

由此可见，多项式 $G_{11j}(z^{-1})$前 j 项正是 $y_1(k)$关于 $u_1(k)$的单位阶跃响应 $g_{110}, g_{111}, \cdots, g_{11j-1}, g_{11j}, \cdots$的前 j 项，而多项式 $G_{12j}(z^{-1})$的前 j 项正是 $y_1(k)$关于 $u_2(k)$的单位阶跃响应 $g_{120}, g_{121}, \cdots, g_{12j-1}, g_{12j}, \cdots$的前 j 项，则有

$$G_{11j}(z^{-1})=g_{110}+g_{111}z^{-1}+\cdots+g_{11j-1}z^{-j+1}+g_{11jj}z^{-j}+\cdots$$

$$G_{12j}(z^{-1})=g_{120}+g_{121}z^{-1}+\cdots+g_{12j-1}z^{-j+1}+g_{12jj}z^{-j}+\cdots \tag{9-17}$$

同单变量采用的方法一样，将式（9-16）的 $\hat{y}_1(k+j)$的值分解成 k 时刻的已知量和未知量两部分，用 $f_1(k+j)$表示已知量，即

$$
\begin{cases}
f_1(k+1) = (\boldsymbol{G}_{111} - g_{110})\Delta u_1(k) + (\boldsymbol{G}_{121} - g_{120})\Delta u_2(k) + \boldsymbol{F}_{11} y_1(k) \\
f_1(k+2) = z(\boldsymbol{G}_{112} - z^{-1} g_{111} - g_{110})\Delta u_1(k) + \\
\qquad\qquad z(\boldsymbol{G}_{122} - z^{-1} g_{121} - g_{120})\Delta u_2(k) + \boldsymbol{F}_{12} y_1(k) \\
\qquad\qquad \vdots \\
f_1(k+n) = z^{n-1}(\boldsymbol{G}_{11n} - z^{-n+1} g_{11n-1} - \cdots - z^{-1} g_{111} + g_{110})\Delta u_1(k) + \\
\qquad\qquad z^{n-1}(\boldsymbol{G}_{12n} - z^{-n+1} g_{12n-1} - \cdots - z^{-1} g_{121} + g_{120})\Delta u_2(k) + \boldsymbol{F}_{1n} y_1(k)
\end{cases}
$$

则式（9-16）可写为

$$
\begin{aligned}
\hat{y}_1(k+1) &= g_{110}\Delta u_1(k) + g_{120}\Delta u_2(k) + f_1(k+1) \\
\hat{y}_1(k+2) &= g_{111}\Delta u_1(k) + g_{110}\Delta u_1(k+1) + g_{121}\Delta u_2(k) + g_{120}\Delta u_2(k+1) + f_1(k+2) \\
&\vdots \\
\hat{y}_1(k+n) &= g_{11n-1}\Delta u_1(k) + g_{11n-2}\Delta u_1(k+1) + \cdots + g_{110}\Delta u_1(k+n-1) + \\
&\quad g_{12n-1}\Delta u_2(k) + g_{12n-2}\Delta u_2(k+1) + \cdots + g_{120}\Delta u_2(k+n-1) + f_1(k+n)
\end{aligned}
$$

将上式写成矩阵形式

$$
\begin{bmatrix} \hat{y}_1(k+1) \\ \hat{y}_1(k+2) \\ \vdots \\ \hat{y}_1(k+n) \end{bmatrix} =
\begin{bmatrix} g_{110} & & & \boldsymbol{0} \\ g_{111} & g_{110} & & \\ \vdots & \vdots & \ddots & \\ g_{11n-1} & g_{11n-2} & \cdots & g_{110} \end{bmatrix}
\begin{bmatrix} \Delta u_1(k) \\ \Delta u_1(k+1) \\ \vdots \\ \Delta u_1(k+n-1) \end{bmatrix}
+ \begin{bmatrix} g_{120} & & & \boldsymbol{0} \\ g_{121} & g_{120} & & \\ \vdots & \vdots & \ddots & \\ g_{12n-1} & g_{12n-2} & \cdots & g_{120} \end{bmatrix}
\begin{bmatrix} \Delta u_2(k) \\ \Delta u_2(k+1) \\ \vdots \\ \Delta u_2(k+n-1) \end{bmatrix}
+ \begin{bmatrix} f_1(k+1) \\ f_1(k+2) \\ \vdots \\ f_1(k+n) \end{bmatrix} \tag{9-18}
$$

即

$$
\hat{\boldsymbol{Y}}_1 = \boldsymbol{G}_{11}\Delta\boldsymbol{U}_1 + \boldsymbol{G}_{12}\Delta\boldsymbol{U}_2 + \boldsymbol{f}_1 \tag{9-19}
$$

其中 $\hat{\boldsymbol{Y}}_1 = [\hat{y}_1(k+1), \hat{y}_1(k+2), \cdots, \hat{y}_1(k+n)]^{\mathrm{T}}$

$\Delta\boldsymbol{U}_1 = [\Delta u_1(k), \Delta u_1(k+1), \cdots, \Delta u_1(k+n-1)]^{\mathrm{T}}$

$\Delta\boldsymbol{U}_2 = [\Delta u_2(k), \Delta u_2(k+1), \cdots, \Delta u_2(k+n-1)]^{\mathrm{T}}$

$\boldsymbol{f}_1 = [f_1(k+1), f_1(k+2), \cdots, f_1(k+n)]^{\mathrm{T}}$

$$
\boldsymbol{G}_{11} = \begin{bmatrix} g_{110} & & & \boldsymbol{0} \\ g_{111} & g_{110} & & \\ \vdots & \vdots & \ddots & \\ g_{11n-1} & g_{11n-2} & \cdots & g_{110} \end{bmatrix}
$$

$$G_{12}=\begin{bmatrix} g_{120} & & & \mathbf{0} \\ g_{121} & g_{120} & & \\ \vdots & \vdots & \ddots & \\ g_{12n-1} & g_{12n-2} & \cdots & g_{120} \end{bmatrix}$$

2）子系统 2

由式（9-13）的子系统 2 模型与丢番图方程

$$1=\boldsymbol{E}_{2j}(z^{-1})\boldsymbol{A}_2(z^{-1})\Delta+z^{-j}\boldsymbol{F}_{2j}(z^{-1})$$

同理可得
$$\hat{\boldsymbol{Y}}_2=\boldsymbol{G}_{21}\Delta\boldsymbol{U}_1+\boldsymbol{G}_{22}\Delta\boldsymbol{U}_2+\boldsymbol{f}_2 \tag{9-20}$$

其中 $\hat{\boldsymbol{Y}}_2=[\hat{y}_2(k+1),\hat{y}_2(k+2),\cdots,\hat{y}_2(k+n)]^{\mathrm{T}}$

$\Delta\boldsymbol{U}_1=[\Delta u_1(k),\Delta u_1(k+1),\cdots,\Delta u_1(k+n-1)]^{\mathrm{T}}$

$\Delta\boldsymbol{U}_2=[\Delta u_2(k),\Delta u_2(k+1),\cdots,\Delta u_2(k+n-1)]^{\mathrm{T}}$

$\boldsymbol{f}_2=[f_2(k+1),f_2(k+2),\cdots,f_2(k+n)]^{\mathrm{T}}$

$$G_{21}=\begin{bmatrix} g_{210} & & & \mathbf{0} \\ g_{211} & g_{210} & & \\ \vdots & \vdots & \ddots & \\ g_{21n-1} & g_{21n-2} & \cdots & g_{210} \end{bmatrix}$$

$$G_{22}=\begin{bmatrix} g_{220} & & & \mathbf{0} \\ g_{221} & g_{220} & & \\ \vdots & \vdots & \ddots & \\ g_{22n-1} & g_{22n-2} & \cdots & g_{220} \end{bmatrix}$$

2. 目标函数和最优控制律

对式（9-11）所表示的 MIMO 模型，采用目标函数

$$J=\sum_{j=1}^{n}[\boldsymbol{y}(k+j)-\boldsymbol{w}(k+j)]^2+\sum_{j=1}^{m}\lambda(j)[\Delta\boldsymbol{u}(k+j-1)]^2 \tag{9-21}$$

其中 $\boldsymbol{y}(k+j)=\begin{bmatrix} y_1(k+j) \\ y_2(k+j) \end{bmatrix},\Delta\boldsymbol{u}(k+j-1)=\begin{bmatrix} \Delta u_1(k+j-1) \\ \Delta u_2(k+j-1) \end{bmatrix}$

$$\boldsymbol{w}(k+j)=\begin{bmatrix} w_1(k+j) \\ w_2(k+j) \end{bmatrix}=\begin{bmatrix} \alpha^j y_1(k)+(1-\alpha^j)y_{r1} \\ \alpha^j y_2(k)+(1-\alpha^j)y_{r2} \end{bmatrix}$$

将 MIMO 系统式（9-11），可以分解为式（9-12）和式（9-13）两个独立的两输入单输出子系统，其相应的输出预测值分别由式（9-19）和式（9-20）求得，下面将 MIMO 的性能指标分解为两个子系统的性能指标。

将式（9-21）写成

$$J=J_1+J_2 \tag{9-22}$$

其中

$$J_1=\sum_{j=1}^{n}[y_1(k+j)-w_1(k+j)]^2+\sum_{j=1}^{m}\lambda(j)[\Delta u_1(k+j-1)]^2 \tag{9-23}$$

$$J_2=\sum_{j=1}^{n}[y_2(k+j)-w_2(k+j)]^2+\sum_{j=1}^{m}\lambda(j)[\Delta u_2(k+j-1)]^2 \tag{9-24}$$

对于子系统式（9-12），相应的子目标函数为式（9-23），输出预测值为式（9-19）。在式（9-19）中，若用上一时刻的$\Delta \boldsymbol{U}_2$值代替 k 时刻的$\Delta \boldsymbol{U}_2$值，即把$\boldsymbol{G}_{12}\Delta \boldsymbol{U}_2$看成已知量。

则式（9-19）可写成

$$\hat{\boldsymbol{Y}}_1=\boldsymbol{G}_{11}\Delta \boldsymbol{U}_1+\bar{\boldsymbol{f}}_1 \qquad (\bar{\boldsymbol{f}}_1=\boldsymbol{G}_{12}\Delta \boldsymbol{U}_2+\boldsymbol{f}_1)$$

若与单变量系统一样，令

$$\boldsymbol{W}_1=[w_1(k+1),w_1(k+2),\cdots,w_1(k+n)]^T$$

则式（9-23）可表示为

$$J_1=(\boldsymbol{Y}_1-\boldsymbol{W}_1)^{\mathrm{T}}(\boldsymbol{Y}_1-\boldsymbol{W}_1)+\lambda\Delta \boldsymbol{U}_1^{\mathrm{T}}\Delta \boldsymbol{U}_1 \tag{9-25}$$

用 $\boldsymbol{Y}_1$ 的最优预测值$\hat{\boldsymbol{Y}}_1$代替 $\boldsymbol{Y}_1$，把 $\boldsymbol{G}_{12}\Delta \boldsymbol{U}_2+\boldsymbol{f}_1$ 看成 $\bar{\boldsymbol{f}}_1$，

并令

$$\frac{\partial J_1}{\partial \Delta \boldsymbol{U}_1}=0$$

可得

$$\Delta \boldsymbol{U}_1=(\boldsymbol{G}_{11}^{\mathrm{T}}\boldsymbol{G}_{11}+\lambda \boldsymbol{I})^{-1}\boldsymbol{G}_{11}^{\mathrm{T}}(\boldsymbol{W}_1-\bar{\boldsymbol{f}}_1)$$

即

$$\Delta \boldsymbol{U}_1=(\boldsymbol{G}_{11}^{\mathrm{T}}\boldsymbol{G}_{11}+\lambda \boldsymbol{I})^{-1}\boldsymbol{G}_{11}^{\mathrm{T}}(\boldsymbol{W}_1-\boldsymbol{G}_{12}\Delta \boldsymbol{U}_2-\boldsymbol{f}_1) \tag{9-26}$$

同理可得

$$\Delta \boldsymbol{U}_2=(\boldsymbol{G}_{22}^{\mathrm{T}}\boldsymbol{G}_{22}+\lambda \boldsymbol{I})^{-1}\boldsymbol{G}_{22}^{\mathrm{T}}(\boldsymbol{W}_2-\boldsymbol{G}_{21}\Delta \boldsymbol{U}_1-\boldsymbol{f}_2) \tag{9-27}$$

当系统已知时，可事先算出 $\boldsymbol{G}_{11}$, $\boldsymbol{G}_{12}$, $\boldsymbol{G}_{21}$, $\boldsymbol{G}_{22}$, $\boldsymbol{f}_1$, $\boldsymbol{f}_2$，然后根据式（9-26）和式（9-27）可得到 $\boldsymbol{U}_1$ 和 $\boldsymbol{U}_2$。

3．参数的辨识

当系统参数未知或具有慢时变时，同以上介绍的单输入单输出系统一样，可将系统辨识与控制策略相结合，构成相应的自校正控制算法。

根据式（9-19），可得 n 个并列预测器为

$$\begin{cases} y_1(k+1)=g_{110}\Delta u_1(k)+g_{120}\Delta u_2(k)+f_1(k+1)+\boldsymbol{E}_{11}\xi_1(k+1) \\ y_1(k+2)=g_{111}\Delta u_1(k)+g_{110}\Delta u_1(k+1)+ \\ \qquad g_{121}\Delta u_2(k)+g_{120}\Delta u_2(k+1)+f_1(k+2)+\boldsymbol{E}_{12}\xi_1(k+2) \\ \qquad \vdots \\ y_1(k+n)=g_{11n-1}\Delta u_1(k)+g_{11n-2}\Delta u_1(k+1)+\cdots+g_{110}\Delta u_1(k+n-1)+ \\ \qquad g_{12n-1}\Delta u_2(k)+g_{12n-2}\Delta u_2(k+1)+\cdots+g_{120}\Delta u_2(k+n-1)+f_1(k+n)+ \\ \qquad \boldsymbol{E}_{1n}\xi_1(k+n) \end{cases}$$

将最后一个方程写成矩阵形式

$$y_1(k+n)=\boldsymbol{X}_1(k)\boldsymbol{\theta}_1(k)+\boldsymbol{E}_{1n}\xi_1(k+n) \tag{9-28}$$

其中

$$\boldsymbol{X}_1(k)=[\Delta u_1(k),\cdots,\Delta u_1(k+n-1),\Delta u_2(k),\cdots,\Delta u_2(k+n-1),1]$$

$$\boldsymbol{\theta}_1(k)=[g_{11n-1},g_{11n-2},\cdots,g_{110},g_{12n-1},g_{12n-2},\cdots,g_{120},f_1(k+n)]^{\mathrm{T}}$$

输出预测值为

$$y_1(k+n/k)=\boldsymbol{X}_1(k)\boldsymbol{\theta}_1(k)$$

或

$$y_1(k/k-n)=\boldsymbol{X}_1(k-n)\boldsymbol{\theta}_1(k) \tag{9-29}$$

同单输入单输出系统一样，根据最小二乘法，可辨识出矩阵 $\boldsymbol{G}_{11}$，$\boldsymbol{G}_{12}$ 的参数，同理可得 $\boldsymbol{G}_{21}, \boldsymbol{G}_{22}$ 的参数。

9.3 仿真研究

9.3.1 单输入单输出系统的仿真研究

1. 跟踪给定值特性

1）最小相位系统

例 9-1 已知系统模型为

$$y(k)-0.496585y(k-1)=0.5u(k-2)+\xi(k)/\Delta$$

取参数：$p=n=6$, $m=2$, $\lambda=0.8$, $\alpha=0.3$, $\lambda_1=1$；RLS 参数初值：$g_{n-1}=1$, $f(k+n)=1$, $p_0=10^5 I$，其余为零；$\xi(k)$为[-0.2,0.2]均匀分布的白噪声，利用附录 A 程序%example9_1.m，可得如图 9-1 所示特性曲线。

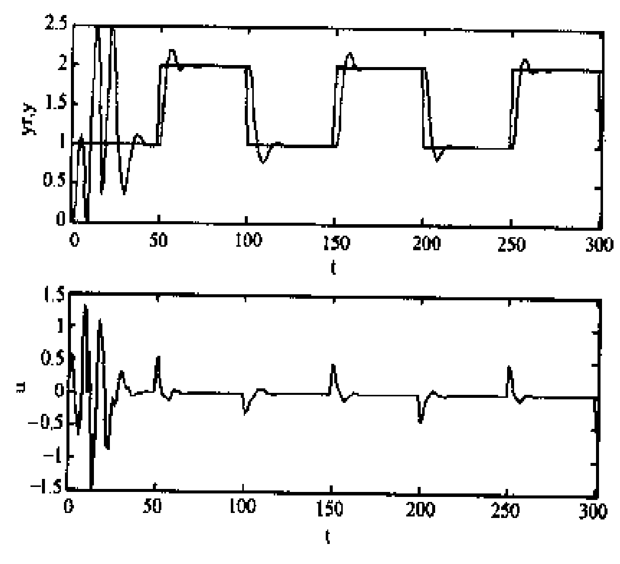


图 9-1 跟踪给定值特性曲线

例 9-2　已知系统模型为

$$y(k)-1.001676y(k-1)+0.241714y(k-2)=0.23589u(k-1)+\xi(k)/\Delta$$

取参数：$p=n=6$, $m=2$, $\lambda=0.5$, $\alpha=0.35$, $\lambda_1=1$；RLS 参数初值：$g_{n-1}=1$, $f(k+n)=1$, $p_0=10^5I$，其余为零；$\xi(k)$为[−0.2,0.2]均匀分布的白噪声，利用附录 A 程序%example9_2.m，可得如图 9-2 所示特性曲线。

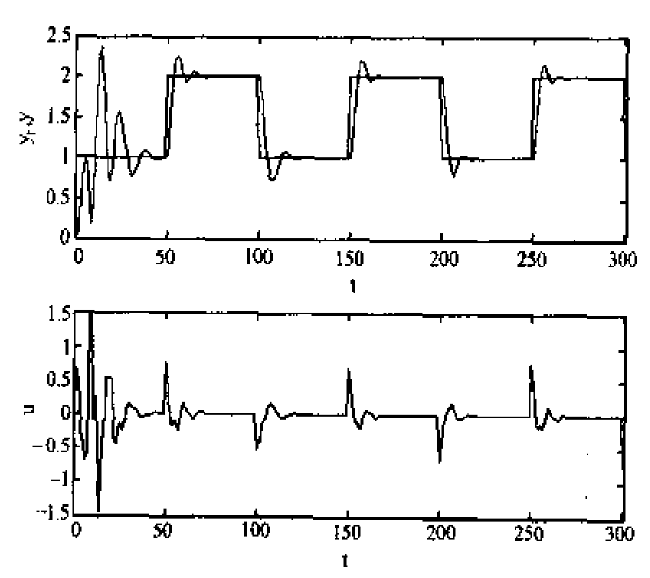


图 9-2　跟踪给定值特性曲线

2）非最小相位系统

例 9-3　已知系统模型为

$$y(k)-1.5y(k-1)+0.7y(k-2)=u(k-1)+1.5u(k-2)+\xi(k)/\Delta$$

用上述非最小相位系统与非线性环节构成非线性控制系统，如图 9-3 所示。

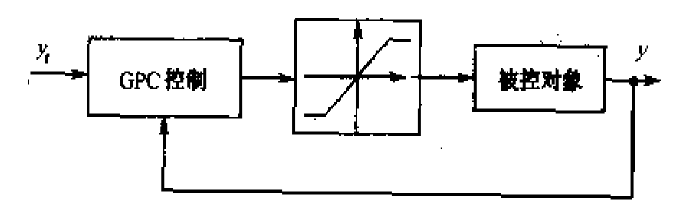


图 9-3　具有非线性特性的非最小相位控制系统

取参数：$p=n=6$, $m=2$, $\lambda=0.8$, $\alpha=0.3$, $\lambda_1=1$；RLS 参数初值：$g_{n-1}=1$, $f(k+n)=1$, $p_0=10^5I$，其余为零；$\xi(k)$为[−0.2,0.2]均匀分布的白噪声，利用附录 A 程序%example9_3.m，可得如图 9-4 所示特性曲线。

2．模型变化时的跟踪给定值特性

1）模型时滞变化时的跟踪特性

例 9-4　已知系统模型为

（1）$y(k)-0.496585y(k-1)=0.5u(k-2)+\xi(k)/\Delta$

（2）$y(k)-0.496585y(k-1)=0.5u(k-3)+\xi(k)/\Delta$

（3）$y(k)-0.496585y(k-1)=0.5u(k-1)+\xi(k)/\Delta$

从第 50 步开始，每 100 步变化一次模型，即 50～150 步采用模型（1）；150～250 步采用模型（2）；250～350 步采用模型（3）。

取参数：$p=n=6$, $m=2$, $\lambda=0.8$, $\alpha=0.3$, $\lambda_1=1$；RLS 参数初值：$g_{n-1}=1, f(k+n)=1, p_0=10^5 I$，其余为零；$\xi(k)$为[−0.2,0.2]均匀分布的白噪声，可得如图 9-5 所示特性曲线。

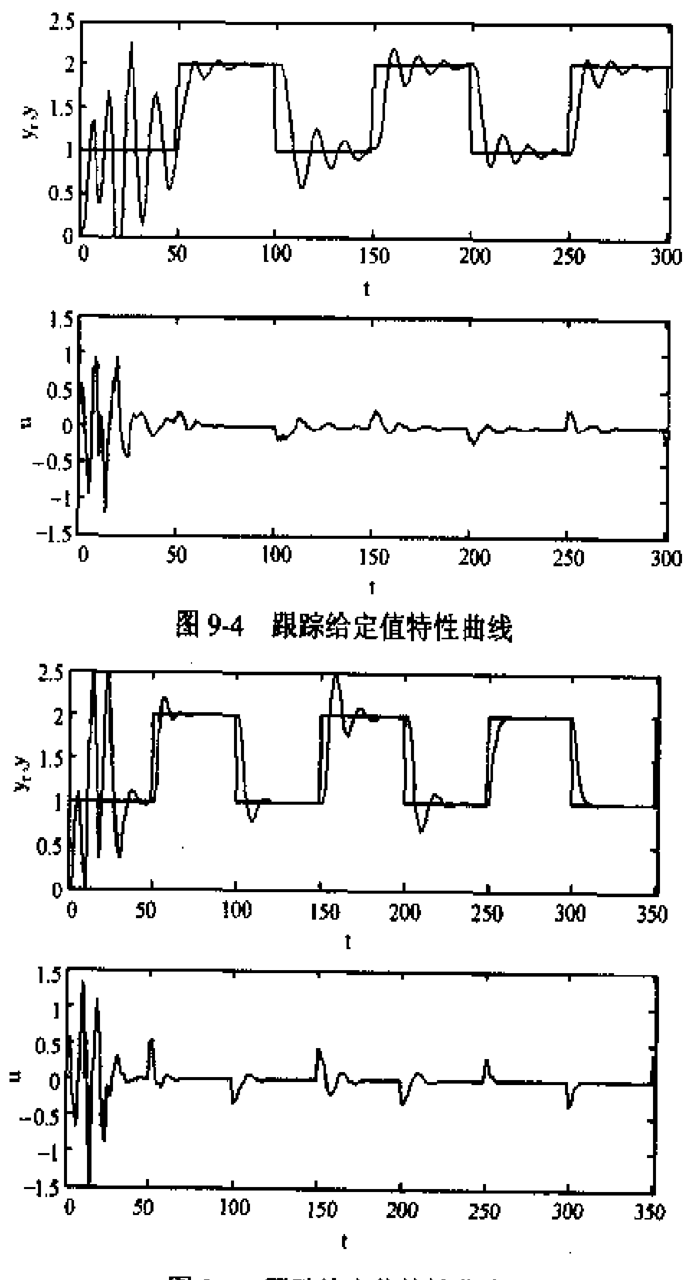


图 9-4　跟踪给定值特性曲线

图 9-5　跟踪给定值特性曲线

2）模型阶次变化时的跟踪特性

例 9-5　已知系统模型为

（1）$y(k)-0.496585y(k-1)=0.5u(k-2)+\xi(k)/\Delta$

（2）$y(k)-1.001676y(k-1)+0.241714y(k-2)=0.23589u(k-1)+\xi(k)/\Delta$

从第 50 步开始，每 100 步变化一次模型，即 50～150 步采用模型（1）；150～300 步采用模型（2）。

取参数：$p=n=6$, $m=2$, $\lambda=0.6$, $\alpha=0.35$, $\lambda_1=1$；RLS 参数初值：$g_{n-1}=1$, $f(k+n)=1$, $p_0=10^5I$，其余为零；$\xi(k)$为[−0.2,0.2]均匀分布的白噪声，可得如图 9-6 所示特性曲线。

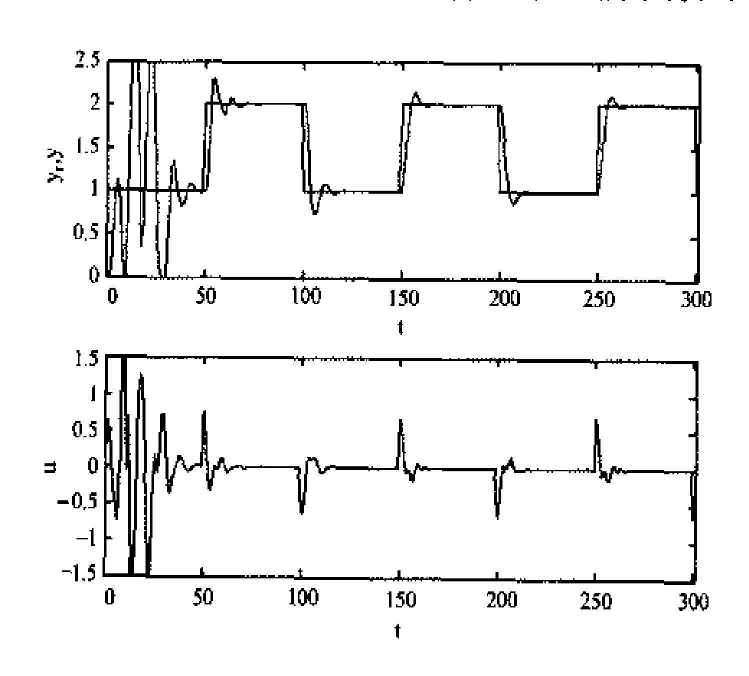


图 9-6　跟踪给定值特性曲线

3）模型阶次和时滞变化时的跟踪特性

例 9-6　已知系统模型为

（1）$y(k)-0.496585y(k-1)=0.5u(k-2)+\xi(k)/\Delta$

（2）$y(k)-1.001676y(k-1)+0.241714y(k-2)=0.23589u(k-1)+\xi(k)/\Delta$

（3）$y(k)-0.496585y(k-1)=0.5u(k-1)+\xi(k)/\Delta$

（4）$y(k)-1.001676y(k-1)+0.24714y(k-2)=0.23589u(k-2)+\xi(k)/\Delta$

从第 50 步开始，每 100 步变化一次模型，即 50～150 步采用模型（1）；150～250 步采用模型（2）；250～350 步采用模型（3）；350～450 步采用模型（4）。

取参数：$p=n=6$, $m=2$, $\lambda=0.6$, $\alpha=0.35$, $\lambda_1=1$；RLS 参数初值：$g_{n-1}=1$, $f(k+n)=1$, $p_0=10^5I$，其余为零；$\xi(k)$为[−0.2,0.2]均匀分布的白噪声，可得如图 9-7 所示特性曲线。

3. 主要参数的改变对系统性能的影响

例 9-7　对例 9-1 模型：取参数：$p=n=6$, $m=1$, $\lambda=0.8$, $\alpha=0.3$, $\lambda_1=1$；RLS 参数初值：$g_{n-1}=1$, $f(k+n)=1$, $p_0=10^5I$，其余为零；$\xi(k)$为[−0.2,0.2]均匀分布的白噪声，可得如图 9-8 所示特性曲线。

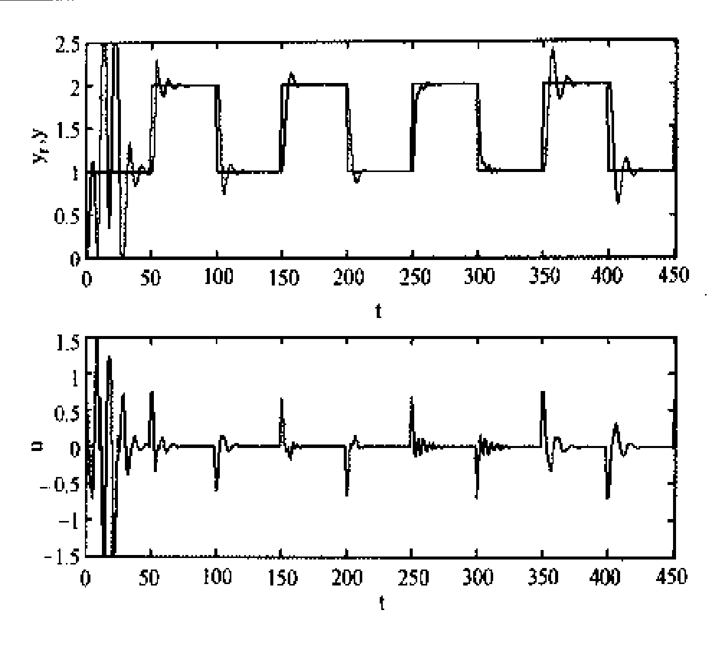


图 9-7 跟踪给定值特性曲线

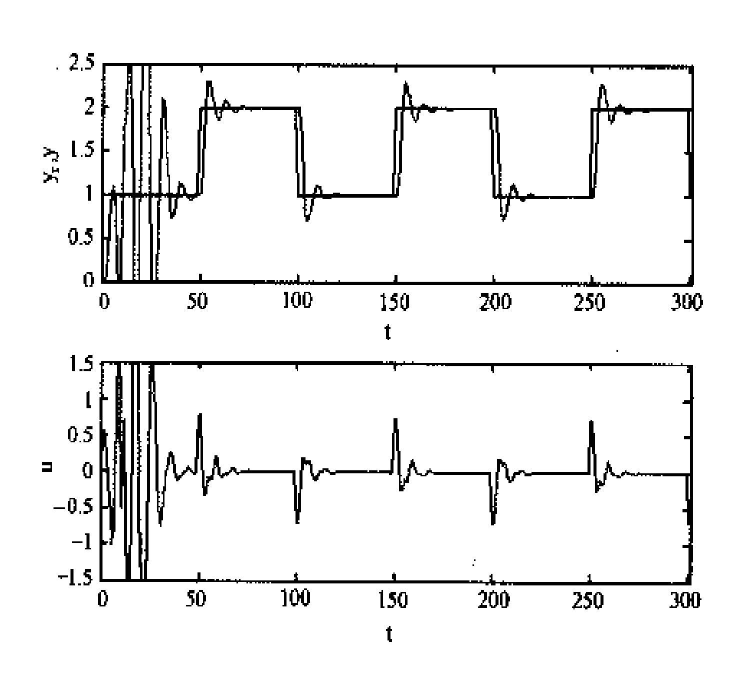


图 9-8 跟踪给定值特性曲线

取参数：$p=n=6, m=2, \lambda=0.6, \alpha=0.3, \lambda_1=1$；RLS 参数初值：$g_{n-1}=1, f(k+n)=1, p_o=10^5 I$，其余为零；$\xi(k)$为[−0.2,0.2]均匀分布的白噪声，可得如图 9-9 所示特性曲线。

例 9-8 对例 9-3 模型：取参数：$p=n=6$, $m=1$, $\lambda=0.8$, $\alpha=0.3$, $\lambda_1=1$：RLS 参数初值：$g_{n-1}=1, f(k+n)=1, p_o=10^5 I$，其余为零；$\xi(k)$为[−0.2,0.2]均匀分布的白噪声，可得如图 9-10 所示特性曲线。

取参数：$p=n=6, m=2, \lambda=0.6, \alpha=0.3, \lambda_1=1$；RLS 参数初值：$g_{n-1}=1, f(k+n)=1, p_o=10^5 I$，其余为零；$\xi(k)$为[−0.2,0.2]均匀分布的白噪声，可得如图 9-11 所示特性曲线。

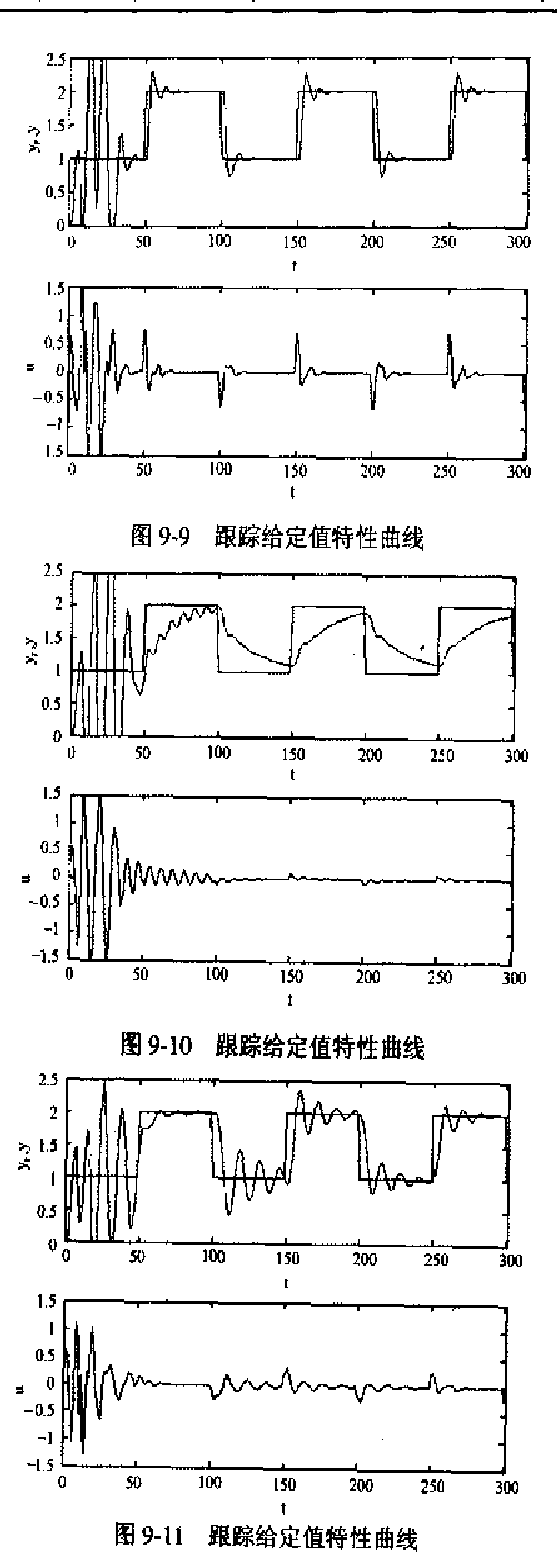


图 9-9　跟踪给定值特性曲线

图 9-10　跟踪给定值特性曲线

图 9-11　跟踪给定值特性曲线

1）m 的改变对系统性能的影响

由图 9-10 与图 9-4 比较可知，当 m 由 1 变为 2 时，系统的性能明显变好，即增大了系统的快速性，但产生振荡和超调。

由图 9-8 与图 9-1 比较，可知当 m 由 2 变为 1 时，系统仍能较好地工作，即对较简单的系统，m 取 1 是可行的。

2）λ的改变对系统性能的影响

由图 9-9 与图 9-1 及图 9-11 与图 9-4 比较可知，当λ由 0.6 变为 0.8 时，控制量明显减少，输出响应速度减慢，但稳定性增强了。

9.3.2 多输入多输出系统的仿真研究

例 9-9 已知系统模型为

$$\begin{bmatrix} 1-0.5z^{-1} & 0 \\ 0 & 1-1.0017z^{-1}+0.2417z^{-2} \end{bmatrix}\begin{bmatrix} y_1(k) \\ y_2(k) \end{bmatrix} = \begin{bmatrix} 0.3z^{-1} & 0.2z^{-1} \\ 0.13z^{-2} & 0.106z^{-1} \end{bmatrix}\begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix} + \begin{bmatrix} \xi_1(k)/\Delta \\ \xi_2(k)/\Delta \end{bmatrix}$$

取参数：$p=n=6, m=2, \lambda=0.8, \alpha=0.3, \lambda_1=1$；RLS 参数初值：$g_{n-1}=1, f(k+n)=1, p_0=10^5I$，其余为零；$\xi(k)$为[−0.2,0.2]均匀分布的白噪声，给定值 y_r 每 50 拍变化一次，利用附录 A 程序%example9_9.m，可得如图 9-12 所示。

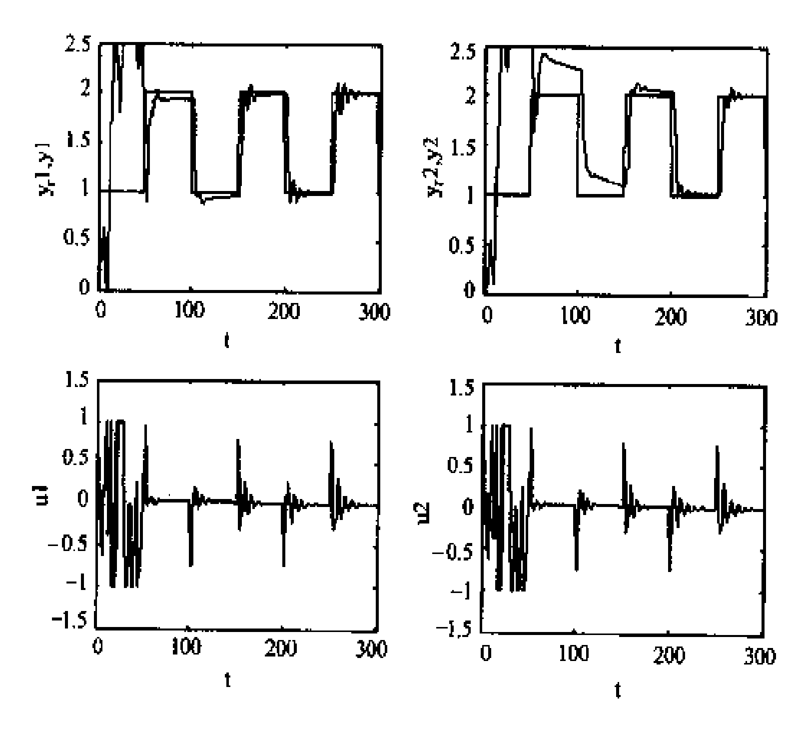


图 9-12 跟踪给定值特性曲线

在递推估计参数时未利用任何关于对象参数的先验知识，即在设定初值时除取 $g_{n-1}=1$，$f(k+n)=1$，$p_o=10^5 I$ 外，其余参数为零，会使控制动作和输出在启动时过度超限。在实际控制时，为改善启动阶段的估计和控制，可首先测出对象的单位阶跃响应系数，以此作为初始参数。

由仿真结果可知，在不需要对象任何先验知识，如模型的阶次、延时时间等情况下，改进的隐式广义预测自校正控制器仍具有较强的适应能力和较好的控制性能，对模型的阶次、时滞和参数的变化都有较强的鲁棒性，在非线性系统中也能得到令人满意的结果。

附录 A　隐式广义预测自校正控制仿真程序清单

```
%example9_1/9_2.m
clear
disp('单变量系统的隐式广义预测控制算法的研究')
disp('广义预测控制算法初始值')
nn=input('时域长度 nn=');
n=input('预测长度 n=');
m=input('控制长度 m=');
t0=input('控制加权系数 λ =');
a=input('柔化系数 α =');
disp('最小二乘公式初始值')
t1=1;                                   %input('遗忘因子 λ 1=');
d1=input('(n+1)阶方阵 P 的形式：0-对角阵, 1-方阵:');
d2=input('(n+1)阶方阵 P 的初始值：0-自动赋值 1e+5, 1-键盘输入:');
if (d1==1)
    if (d2==1)
        P=input('在方括号[ ]中，输入(n+1)阶方阵 P 的值:');
    else
        P=(1e+5)*ones(n+1);
    end
else
    if (d2==1)
        PP=input('在方括号[ ]中，输入(n+1)阶对角方阵 P 对角线上的值:');
        P=diag(PP);
    else
        P=(1e+5)*eye(n+1);
    end
end
%参数初始值
uuu=0;yyy=0;
```

```
uu=zeros(n,1);u=zeros(m,1);
yy=zeros(n,1);y1=zeros(n,1);
Q=zeros(n+1,1);Q(1,1)=1;Q(n+1,1)=1;
%产生周期为 100，时间为 T，幅值为 1 的方波信号的给定值
T=300;[yr0,t]=gensig('square',100,T,1);
d3=input('输出曲线是否去掉前 100 步: 0-不, 1-去掉:');
nm=length(t);                          %确定循环次数
for ij=2:nm
yr=yr0(ij)+1;    %产生周期为 100，时间为 T, 幅值在 1 和 2 之间变化的方波信号的给定值
%根据系统模型，计算 k 时刻的输出值 y(k)
y=1.496585*yy(n,1)-0.496585*yy(n-1,1)+0.5*uu(n-1,1);
          %(nn=6;n=6;m=2;t0=0.8;a=0.3;t1=1)                    %example9-1 仿真模型
%y=2.001676*yy(n,1)-1.24339*yy(n-1,1)+0.24171*yy(n-2,1)+0.23589*uu(n,1);  %example9-2 仿真模型
%产生均匀分布的白噪声
a9=0;
for i=1:1
    a9=a9+rand;
end
a8=0.01*(a9-6);
%保存 k 时刻及以前的 n 个输出值 y(k),y(k-1),…,y(k-n), 以供模型运算
for i=1:n-1
    yy(i,1)=yy(i+1,1);
end
yy(n,1)=y;
yyy=[yyy;y];                    %保存各 k 时刻的 nm 个输出量以便绘图
%根据最小二乘公式，由 y(k)计算 G 阵的各元素值 g0,g1,…,gn
for i=1:n
    X(1,i)=uu(i,1);
end
X(1,n+1)=1;
K=P*X'*inv(t1+X*P*X');
P=(eye(n+1)-K*X)*P/t1;
Q=Q+K*(y-X*Q);
%根据元素值 g0,g1,…,gn, 求 G 阵
```

```
for j=1:m
    for i=n:-1:j
        i1=n-i+j;
        G(i1,j)=Q(i,1);
    end
end
%求 nn 维 y0 向量（y1 为上一时刻的 y0 向量）
y0(1:nn-1,1)=y1(2:nn,1);y0(nn,1)=y1(nn,1);
y0=y0+(y-y1(1,1));
for i=1:n
    y1(i,1)=y0(i,1)+u(1,1)*Q(n+1-i,1);
end
for i=n+1:nn
    y1(i,1)=y1(n,1);
end
%根据 y0, 求 n 维 f 向量 f(k+1),…,f(k+n)
f(1:n,1)=y0(1:n,1);
%由当前 k 时刻的输出值 y(k)和给定值 yr, 求 k 时刻以后的 n 个参考轨迹 w(k+1),…,w(k+n)
w=a*y+(1-a)*yr;
for i=2:n
    w=[w;a^i*y+(1-a^i)*yr];
end
%计算 k 时刻及以后的 m 个控制增量 Du(k),…,Du(k+m)
u=inv(G'*G+t0*eye(m))*G'*(w-f);
%保存 k 时刻及以后的 n 个控制增量, 以供模型运算
for i=1:n-1
    uu(i,1)=uu(i+1,1);
end
uu(n,1)=u(1,1);
uuu=[uuu;u(1,1)];                  %保存各 k 时刻的 nm 个控制增量以便绘图
%控制量限幅
if(u(1,1)>1)
    u(1,1)=1;
end
```

```
        if(u(1,1)<-1)
            u(1,1)=-1;
        end
    end
    %绘制给定值、输出值和控制增量曲线
    if (d3==1)
        %绘制去掉前100步的给定值、输出值和控制增量曲线
        yyy1(1:(T-100),1)=yyy(101:T,1);uuu1(1:(T-100),1)=uuu(101:T,1);
        t1(1:(T-100),1)=t(101:T,1)-100;yr01(1:(T-100),1)=yr0(101:T,1);
        subplot(2,1,1):plot(t1,(yr01+1),t1,yyy1);
        axis([0,nm-100,0,2.5]);xlabel('t');ylabel('yr,y')
        subplot(2,1,2):plot(t1,uuu1);
        axis([0,nm-100,-1.5,1.5]);xlabel('t');ylabel('u')
    else
        %绘制完整的给定值、输出值和控制增量曲线
        subplot(2,1,1):plot(t,(yr0+1),t,yyy);
        axis([0,nm,0,2.5]);xlabel('t');ylabel('yr,y')
        subplot(2,1,2):plot(t,uuu);
        axis([0,nm,-1.5,1.5]);xlabel('t');ylabel('u')
    end
```

%example9_3.m

```
clear
disp('单变量系统的隐式广义预测控制算法的研究')
disp('广义预测控制算法初始值')
nn=input('时域长度 nn=');
n=input('预测长度 n=');
m=input('控制长度 m=');
t0=input('控制加权系数λ=');
a=input('柔化系数α=');
disp('最小二乘公式初始值')
t1=1;                                   %input('遗忘因子λ1=');
d1=input('(n+1)阶方阵P的形式：0-对角阵，1-方阵:');
```

```
d2=input('(n+1)阶方阵 P 的初始值: 0-自动赋值 1e+5, 1-键盘输入:');
if (d1==1)
    if (d2==1)
        P=input('在方括号[ ]中, 输入(n+1)阶方阵 P 的值:');
    else
        P=(1e+5)*ones(n+1);
    end
else
    if (d2==1)
        PP=input('在方括号[ ]中, 输入(n+1)阶对角方阵 P 对角线上的值:');
        P=diag(PP);
    else
        P=(1e+5)*eye(n+1);
    end
end
%参数初始值
uuu=0;yyy=0;
uu=zeros(n,1);u=zeros(m,1);
yy=zeros(n,1);y1=zeros(n,1);
Q=zeros(n+1,1);Q(1,1)=1;Q(n+1,1)=1;
%产生周期为 100, 时间为 T, 幅值为 1 的方波信号的给定值
T=300;[yr0,t]=gensig('square',100,T,1);
d3=input('输出曲线是否去掉前 100 步: 0-不, 1-去掉:');
nm=length(t);                              %确定循环次数
for ij=2:nm
   yr=yr0(ij)+1;    %产生周期为 100, 时间为 T, 幅值在 1 和 2 之间变化的方波信号的给定值
   %根据系统模型, 计算 k 时刻的输出值 y(k)
   y=2.5*yy(n,1)-2.2*yy(n-1,1)+0.7*yy(n-2,1)+(uu(n,1)+1.5*uu(n-1,1))/12.5;
                    %(nn=6;n=6;m=2;t0=0.8;a=0.5;t1=1)
   %产生均匀分布的白噪声
   a9=0;
   for i=1:1
       a9=a9+rand;
   end
```

```
a8=0.01*(a9-6);
%保存k时刻及以前的n个输出值y(k),y(k-1),…,y(k-n), 以供模型运算
for i=1:n-1
     yy(i,1)=yy(i+1,1);
end
yy(n,1)=y;
yyy=[yyy;y];                              %保存各k时刻的nm个输出量以便绘图
%根据最小二乘公式，由y(k)计算G阵的各元素值g0,g1,…,gn
for i=1:n
     X(1,i)=uu(i,1);
end
X(1,n+1)=1;
K=P*X'*inv(t1+X*P*X');
P=(eye(n+1)-K*X)*P/t1;
Q=Q+K*(y-X*Q);
%根据元素值g0,g1,…,gn, 求G阵
for j=1:m
     for i=n:-1:j
          i1=n-i+j;
          G(i1,j)=Q(i,1);
     end
end
%根据y1（y1为上一时刻的y0向量）,求n维f向量f(k+1),…,f(k+n)
e=y-y1(1,1);
f(1:n-1,1)=y1(2:n,1)+e;f(n,1)=y1(n,1)+e;
y1=f+G*u;
%由当前k时刻的输出值y(k)和给定值yr, 求k时刻以后的n个参考轨迹w(k+1),…,w(k+n)
w=a*y+(1-a)*yr;
for i=2:n
     w=[w;a^i*y+(1-a^i)*yr];
end
%计算k时刻及以后的m个控制增量Du(k),…,Du(k+m)
u=inv(G'*G+t0*eye(m))*G'*(w-f);
%保存k时刻及以后的n个控制增量, 以供模型运算
```

```
        for i=1:n-1
            uu(i,1)=uu(i+1,1);
        end
        uu(n,1)=u(1,1);
        uuu=[uuu;u(1,1)]; %保存各 k 时刻的 nm 个控制增量以便绘图
        %控制量限幅
        if(u(1,1)>0.5*yr)
            u(1,1)=0.5*yr;
        end
        if(u(1,1)<-0.5*yr)
            u(1,1)=-0.5*yr;
        end
end
%绘制给定值、输出值和控制增量曲线
if (d3==1)
        %绘制去掉前 100 步的给定值、输出值和控制增量曲线
        yyy1(1:(T-100),1)=yyy(101:T,1);uuu1(1:(T-100),1)=uuu(101:T,1);
        t1(1:(T-100),1)=t(101:T,1)-100;yr01(1:(T-100),1)=yr0(101:T,1);
        subplot(2,1,1):plot(t1,(yr01+1),t1,yyy1);
        axis([0,nm-100,0,2.5]);xlabel('t');ylabel('yr,y')
        subplot(2,1,2):plot(t1,uuu1);
        axis([0,nm-100,-1.5,1.5]);xlabel('t');ylabel('u')
else
        %绘制完整的给定值、输出值和控制增量曲线
        subplot(2,1,1):plot(t,(yr0+1),t,yyy);
        axis([0,nm,0,2.5]);xlabel('t');ylabel('yr,y')
        subplot(2,1,2):plot(t,uuu);
        axis([0,nm,-1.5,1.5]);xlabel('t');ylabel('u')
end
```

```
%example9_9.m
clear
disp('多变量系统的隐式广义预测控制算法的研究')
```

```
disp('广义预测控制算法初始值')
nn=input('时域长度 nn=');
n=input('预测长度 n=');
m=input('控制长度 m=');
t0=input('控制加权系数 λ =');
a=input('柔化系数 α =');
disp('最小二乘公式初始值')
t1=1;                                   %input('遗忘因子 λ 1=');
d1=input('(2*n+1)阶方阵 P 的形式: 0-对角阵, 1-方阵:');
d2=input('(2*n+1)阶方阵 P 的初始值: 0-自动赋值 1e+5, 1-键盘输入:');
if (d1==1)
    if (d2==1)
        P=input('在方括号[ ]中, 输入(2*n+1)阶方阵 P 的值:');
    else
        P=(1e+5)*ones(2*n+1);
    end
else
    if (d2==1)
        PP=input('在方括号[ ]中, 输入(2*n+1)阶对角方阵 P 对角线上的值:');
        P=diag(PP);
    else
        P=(1e+5)*eye(2*n+1);
    end
end
%参数初始值
uuu1=0;yyy1=0;uuu2=0;yyy2=0;
uu1=zeros(n,1);u1=zeros(m,1);uu2=zeros(n,1);u2=zeros(m,1);
yy1=zeros(n,1);y11=zeros(n,1);yy2=zeros(n,1);y12=zeros(n,1);
Q1=zeros(2*n+1,1);Q1(1,1)=1;Q1(n+1,1)=1;Q1(2*n+1,1)=1;
Q2=Q1;
%产生周期为 100, 时间为 T, 幅值为 1 的方波信号的给定值
T=300;[yr0,t]=gensig('square',100,T,1);
d3=input('输出曲线是否去掉前 100 步: 0-不, 1-去掉:');
nm=length(t);                           %确定循环次数
```

```
for ij=2:nm
    yr1=yr0(ij)+1;%产生周期为 100，时间为 T, 幅值在 1 和 2 之间变化的方波信号的给定值
    yr2=yr1;
    %根据系统模型，计算 k 时刻的输出值 y1(k)和 y2(k)
    y1=(1+exp(-.7))*yy1(n,1)-exp(-.7)*yy1(n-1,1)+0.3*uu1(n,1)+0.2*uu2(n-1,1);
    y2=2.0017*yy2(n,1)-1.2434*yy2(n-1,1)+.2417*yy2(n-2,1)+.13*uu1(n-1,1)+.1059*uu2(n,1);
                                        %(nn=n=6;m=2;t0=0.8;a=0.3;t1=1)
    %保存 k 时刻及以前的 n 个输出值 y(k),y(k-1),…,y(k-n), 以供模型运算
    for i=1:n-1
        yy1(i,1)=yy1(i+1,1);yy2(i,1)=yy2(i+1,1);
    end
    yy1(n,1)=y1;yy2(n,1)=y2;
    yyy1=[yyy1;y1];yyy2=[yyy2;y2];%保存各 k 时刻的 nm 个输出量以便绘图
    %根据最小二乘公式，由 y1(k)和 y2(k)计算 G11,G12,G21 和 G22 阵的各元素值 g0,g1,…,gn
    for i=1:n
        X(1,i)=uu1(i,1);X(1,i+n)=uu2(i,1);
    end
    X(1,2*n+1)=1;
    K=P*X'*inv(t1+X*P*X');
    P=(eye(2*n+1)-K*X)*P/t1;
    Q1=Q1+K*(y1-X*Q1);Q2=Q2+K*(y2-X*Q2);
    %根据元素值 g0,g1,...,gn,求 G11,G12,G21 和 G22 阵
    for j=1:m
        for i=n:-1:j
            i1=n-i+j;
            G11(i1,j)=Q1(i,1);G12(i1,j)=Q1(i+n,1);
            G21(i1,j)=Q2(i,1);G22(i1,j)=Q2(i+n,1);
        end
    end
    %for i=1:n,G12(i,2)=0;G21(i,2)=0;end
    %求 nn 维 y01,y02 向量（y11 和 y12 为上一时刻的 y01 和 y02 向量）
    e1=y1-y11(1,1);e2=y2-y12(1,1);
    y01(1:nn-1,1)=y11(2:nn,1);y01(nn,1)=y11(nn,1);y01=y01+e1;
    y02(1:nn-1,1)=y12(2:nn,1);y02(nn,1)=y12(nn,1);y02=y02+e2;
```

```
        for i=1:n
            y11(i,1)=y01(i,1)+u1(1,1)*Q1(n+1-i,1)+u2(1,1)*Q1(2*n+1-i,1);
            y12(i,1)=y02(i,1)+u2(1,1)*Q2(n+1-i,1)+u1(1,1)*Q2(2*n+1-i,1);
        end
        for i=n+1:nn
            y11(i,1)=y11(n,1);y12(i,1)=y12(n,1);
        end
        %根据y01,y02求n维f1,f2向量
        f1(1:n,1)=y01(1:n,1);
        f2(1:n,1)=y02(1:n,1);
        %保存k时刻及以后的n个控制增量，以供模型运算
        for i=1:n-1
            uu1(i,1)=uu1(i+1,1);uu2(i,1)=uu2(i+1,1);
        end
        uu1(n,1)=u1(1,1);uu2(n,1)=u2(1,1);
        uuu1=[uuu1;u1(1,1)];uuu2=[uuu2;u2(1,1)];   %保存各k时刻的nm个控制增量以便绘图
        %由当前k时刻的输出值y(k)和给定值yr，求k时刻以后的n个参考轨迹w(k+1),…,w(k+n)
        w1=a*y1+(1-a)*yr1;w2=a*y2+(1-a)*yr2;
        for i=2:n
            w1=[w1;a^i*y1+(1-a^i)*yr1];
            w2=[w2;a^i*y2+(1-a^i)*yr2];
        end
        %计算k时刻及以后的m个控制增量Du(k),…,Du(k+m),
        u1=inv(G11'*G11+t0*eye(m))*G11'*(w1-G12*u2-f1);
        u2=inv(G22'*G22+t0*eye(m))*G22'*(w2-G21*u1-f2);
        %控制量限幅
        if(u1(1,1)>1),u1(1,1)=1;end
        if(u1(1,1)<-1),u1(1,1)=-1;end
        if(u2(1,1)>1),u2(1,1)=1;end
        if(u2(1,1)<-1),u2(1,1)=-1;end
    end
    %绘制给定值、输出值和控制增量曲线
    if (d3==1)
        %绘制去掉前100步的给定值、输出值和控制增量曲线
```

```
        yyyy1(1:(T-100),1)=yyy1(101:T,1);uuuu1(1:(T-100),1)=uuu1(101:T,1);
        yyyy2(1:(T-100),1)=yyy2(101:T,1);uuuu2(1:(T-100),1)=uuu2(101:T,1);
        t1(1:(T-100),1)=t(101:T,1)-100;yr01(1:(T-100),1)=yr0(101:T,1);
        subplot(2,2,1):plot(t1,(yr01+1),t1,yyyy1);
        axis([0,nm-100,0,2.5]);xlabel('t');ylabel('yr1,y1')
        subplot(2,2,3):plot(t1,uuuu1);
        axis([0,nm-100,-1.5,1.5]);xlabel('t');ylabel('u1')
        subplot(2,2,2):plot(t1,(yr01+1),t1,yyyy2);
        axis([0,nm-100,0,2.5]);xlabel('t');ylabel('yr2,y2')
        subplot(2,2,4):plot(t1,uuuu2);
        axis([0,nm-100,-1.5,1.5]);xlabel('t');ylabel('u2')
else
        %绘制完整的给定值、输出值和控制增量曲线
        subplot(2,2,1):plot(t,(yr0+1),t,yyy1);
        axis([0,nm,0,2.5]);xlabel('t');ylabel('yr1,y1')
        subplot(2,2,3):plot(t,uuu1);
        axis([0,nm,-1.5,1.5]);xlabel('t');ylabel('u1')
        subplot(2,2,2):plot(t,(yr0+1),t,yyy2);
        axis([0,nm,0,2.5]);xlabel('t');ylabel('yr2,y2')
        subplot(2,2,4):plot(t,uuu1);
        axis([0,nm,-1.5,1.5]);xlabel('t');ylabel('u2')
end
```

附录 B　MATLAB 函数一览表

函 数 名	功 能
A	
abcdchkm( )	检查状态空间矩阵 A,B,C,D 的维数一致性
adapt( )	神经网络自适应训练函数
adaptwb( )	神经网络的权值和阈值自适应函数
adaptwh( )	对线性神经网络进行在线自适应训练
addmd( )	向 MPC 对象添加一个或多个测量扰动
addmf( )	添加模糊语言变量的隶属度函数
addmod( )	将两个开环 MPC 模型连接构成闭环模型
addrule( )	向模糊推理系统添加模糊规则函数
addumd( )	向 MPC 对象添加一个或多个未测量扰动
addvar( )	添加模糊语言变量
anfis( )	模糊神经系统的建模函数
anfisedit	模糊神经推理系统的图形界面工具
appmod( )	用两个 MPC 系统模型构成增广系统模型
autosc( )	矩阵或向量的自动归一化
B	
barerr( )	绘制误差的直方图
C	
c2dmp( )	连续系统离散化
cmpc( )	输入/输出受限的模型预测控制器设计与仿真
compet( )	竞争传输函数
cp2dp( )	将连续型多项式传递函数转换为离散型多项式传递函数
D	
d2cmp( )	将离散系统转换为连续函数
dantzgmp( )	求解二次规划问题
dareiter( )	求解离散 Riccati 方程
defuzz( )	执行输出去模糊化函数
deltalin( )	Purelin 神经元的 delta 函数
deltalog( )	Logsig 神经元的 delta 函数
deltatan( )	Tansig 神经元的 delta 函数
dimpulsm( )	生成离散系统的脉冲响应
dist( )	计算矢量间的距离函数
dlqe2( )	计算离散系统的状态估计器增益矩阵
dlsimm( )	离散系统仿真
dotprod( )	权值点积函数
dsigmf( )	计算两个 sigmiod 隶属度函数之和

续表

函 数 名	功 能
E	
errsurf( )	计算误差曲面
evalfis( )	执行模糊推理计算函数
F	
fcm( )	模糊 C-均值聚类函数
fuzzy	启动模糊推理系统编辑器
G	
gauss2mf( )	建立双边高斯型隶属度函数
gaussmf( )	建立高斯型隶属度函数
gbellmf( )	建立一般的钟型隶属度函数
genfis1( )	采用网格分割方式生成模糊推理系统函数
genfis2( )	基于减法聚类的模糊推理系统建模函数
gensim( )	对一个网络生成其模块化描述
gensurf( )	生成模糊推理系统的输出曲面并显示函数
getfis( )	获得模糊推理系统的特性数据
H	
hardlim( )	硬限幅传输函数
hardlims( )	对称硬限幅传输函数
I	
imp2step( )	由 MISO 脉冲响应模型生成 MIMO 阶跃响应模型
ind2vec( )	将下标矢量变换成单值矢量组
init ( )	初始化一个神经网络
initc( )	初始化竞争神经网络
initelm( )	对 Elman 神经网络进行初始化
initff( )	对 BP 神经网络进行初始化
initlay( )	神经网络某一层的初始化函数
initlin( )	线性神经网络的初始化函数
initlvq( )	初始化 LVQ 神经网络
initp( )	对感知机神经网络进行初始化
initsm( )	初始化自组织特征映射网络
initwb( )	神经网络某一层的权值和阈值初始化函数
initzero( )	将权值设置为零的初始化函数
L	
learnbp( )	BP 学习规则
learnbpm( )	含动量规则的快速 BP 学习规则
learnh ( )	Hebb 权值学习规则函数

续表

函 数 名	功 能
	L
learnhd( )	衰减的 Hebb 权值学习规则函数
learnis ( )	Instar 权值学习规则函数
learnk ( )	Kohonen 权值学习规则函数
learnlm( )	Levenberg-Marguardt 学习规则
learnlvq( )	LVQ 神经网络学习函数
learnos ( )	Outstar 权值学习规则函数
learnp( )	感知机的学习函数
learnpn( )	标准化感知机的学习函数
learnsom( )	自组织特征映射权值学习规则函数
learnwh( )	Widrow-hoff 的学习函数
linkdist( )	Link 距离权值函数
logsig( )	对数 S 型(Log-Sigmoid)传输函数
	M
mae( )	平均绝对误差性能函数
mandist( )	Manhattan 距离权值函数
maxlinlr( )	计算线性层的最大学习速率
mf2mf( )	隶属度函数间的参数转换
midpoit ( )	中点权值初始化函数
mlr( )	利用多变量线性回归计算 MISO 脉冲响应模型
mod2frsp( )	计算系统（MPC 状态空间模型）的频率响应
mod2mod( )	改变 MPC 状态空间模型的采样周期
mod2ss( )	将 MPC 状态空间模型转换为通用状态空间模型
mod2step( )	将 MPC 状态空间模型转换为 MPC 阶跃响应模型
mpcaugss( )	增广状态空间模型
mpccl( )	计算模型预测控制系统的闭环模型
mpccon( )	输入/输出不受限的模型预测控制器设计
mpcinfo( )	输出系统表示的矩阵类型和属性
mpcparal( )	将两个状态空间模型并联
mpcsim( )	模型预测闭环控制系统的仿真（输入/输出不受限）
mpcstair( )	生成阶梯型控制变量
mse( )	均方差性能函数
	N
nbdist( )	用矢量距离表示的领域矩阵
nbgrid( )	用栅格距离表示的领域矩阵
nbman( )	用 Manhattan 距离表示的领域矩阵

续表

函 数 名	功 能
	N
negdist( )	对输入矢量进行加权计算
netprod( )	网络输入的积函数
netsum( )	计算网络输入矢量和
newc( )	建立一个竞争神经网络
newcf( )	生成一个前向层叠 BP 网络
newelm( )	生成一个 Elman 神经网络
newff( )	生成一个前馈 BP 网络
newfftd( )	生成一个前馈输入延时 BP 网络
newfis( )	创建新的模糊推理系统
newgrnn( )	新建一个广义回归径向基神经网络
newhop( )	生成一个 Hopfield 回归网络
newlin( )	新建一个线性层
newlind( )	设计一个线性层
newlvq( )	建立一个 LVQ 神经网络函数
newp( )	生成一个感知机
newpnn( )	新建一个概率径向基神经网络
newrb( )	新建一个径向基神经网络
newrbe( )	新建一个严格的径向基神经网络
newsom( )	创建一个自组织特征映射神经网络
nlcmpc( )	Simulink 块 nlcmpc 对应的 S 函数
nlmpcsim( )	Simulink 块 nlmpcsim 对应的 S 函数
nngenc( )	产生一定类别的样本向量
nntool	启动神经网络编辑器的图形界面
normprod( )	规范点积权值函数
nwlog( )	对 Logsig 神经元产生 Nguyen-Midrow 随机数
	P
paramod( )	将两个 MPC 系统模型并联
parpart( )	分割参数用于 Simulink 仿真
parsrule( )	解析模糊规则函数
pimf( )	建立π型隶属度函数
plotall( )	绘制系统仿真的输入/输出曲线（一个图形窗口）
ploteach( )	在多个图形窗口分别绘制系统的输入/输出仿真曲线
plotep( )	在误差曲面图上绘制权值和阈值的位置
ploterr( )	绘制误差平方和对训练次数的曲线
plotes( )	绘制误差曲面图

续表

函 数 名	功 能
	P
plotfis( )	图形显示模糊推理系统的输入/输出特性
plotfrsp( )	绘制系统的频率响应波特图
plotmf( )	绘制隶属度函数曲线
plotpc( )	在已绘制的图上加分类线
plotpv( )	在坐标图上绘出样本点
plotsm( )	绘制竞争网络的权值矢量
plotsom( )	绘制自组织特征映射网络的权值矢量
plotstep( )	绘制系统阶跃响应模型的曲线
plotvec( )	用不同的颜色画矢量函数
plsr( )	利用部分最小二乘方回归方法计算 MISO 脉冲响应模型
poly2tfd( )	将通用传递函数模型转换为 MPC 传递函数模型
psigmf( )	计算两个 Sigmiod 隶属度函数之积
purelin( )	线性传输函数
	R
radbas( )	径向基传输函数
readfis( )	从磁盘读出存储的模糊推理系统
rescal( )	由归一化的数据生成原数据
rmmf( )	删除隶属度函数
rmvar( )	删除模糊语言变量
	S
satlin( )	饱和线性传输函数
satlins( )	饱和对称线性传输函数
scal( )	根据指定的均值和标准差归一化矩阵
scmpc( )	输入/输出有约束的状态空间模型预测控制器设计
sermod( )	将两个 MPC 系统模型串联
setfis( )	设置模糊推理系统的特性
showfis( )	显示添加注释了的模糊推理系统
showrule( )	显示模糊规则函数
sigmf( )	建立 Sigmiod 型的隶属度函数
sim( )	神经网络仿真函数
simuc( )	仿真竞争神经网络
simuelm( )	对 Elman 神经网络进行仿真
simuff( )	对 BP 神经网络进行仿真
simuhop( )	仿真一个 Hopfield 回归网络
simulin( )	对线性神经网络进行仿真

续表

函 数 名	功 能
S	
simulvq( )	仿真LVQ神经网络
simup( )	对感知机神经网络进行仿真
simurb( )	径向基神经网络仿真函数
simusm( )	仿真自组织特征映射网络
smpccl( )	计算输入/输出无约束的模型预测闭环控制系统模型
smpccon( )	输入/输出无约束的状态空间模型预测控制器设计
smpcest( )	状态估计器设计
smpcgain( )	计算系统（MPC状态空间模型）的稳态增益矩阵
smpcpole( )	计算系统（MPC状态空间模型）的极点
smpcsim( )	模型预测闭环控制系统仿真
solvehop( )	设计一个Hopfield回归网络
solvelin( )	设计一个线性神经网络
solverb( )	设计一个径向基神经网络
solverbe( )	设计一个精确径向基神经网络
ss2mod( )	将通用状态空间模型转换为MPC状态空间模型
ss2step( )	将通用状态空间模型转换为MPC阶跃响应模型
ss2tf2( )	将状态空间模型转换为传递函数
sse( )	误差平方和性能函数
subclust( )	减法聚类函数
sumsqr( )	计算误差平方和
svdfrsp( )	计算频率响应的奇异值
T	
tansig( )	双曲正切S型（Tan-Sigmoid）传输函数
tf2ssm( )	将传递函数转换为状态空间模型
tfd2mod( )	将MPC传递函数模型转换为MPC状态空间模型
tfd2step( )	将MPC传递函数模型转换为MPC阶跃响应模型
th2mod( )	将Theta格式模型转换为MPC状态空间模型
train( )	神经网络训练函数
trainbp( )	利用BP算法训练前向网络
trainbpx( )	利用快速BP算法训练前向网络
trainc( )	训练竞争神经网络
trainelm( )	训练Elman神经网络的权值和阈值
trainlm( )	利用Levenberg-Marguardt规则训练前向网络
trainlvq( )	训练LVQ神经网络
trainp( )	训练感知机神经网络的权值和阈值

续表

函　数　名	功　　能
T	
trainpn( )	训练标准化感知机的权值和阈值
trainsm( )	利用 Kohonen 规则训练自组织特征映射网络
trainwb( )	神经网络的权值和阈值训练函数
trainwh( )	对线性神经网络进行离线训练
trapmf( )	建立梯形隶属度函数
trimf( )	建立三角型隶属度函数
V	
validmod( )	利用新的数据检验 MISO 脉冲响应模型
vec2ind ( )	将单值矢量组变换成下标矢量
W	
writefis( )	保存模糊推理系统
wrtreg( )	生成用于线性回归计算的数据矩阵
Z	
zmf( )	建立 Z 型隶属度函数

附录C　MATLAB 函数分类索引

1．神经网络控制工具箱函数

1）感知机神经网络函数

mae( ); hardlim( ); hardlims( ); plotpv( ); plotpc( ); initp( ); trainp( ); trainpn( ); simup( ); learnp( ); learnpn( );newp( ); netsum( ); train( ); adapt( ); sim( ); init ( )

2）线性神经网络函数

purelin( ); initlin( ); solvelin( ); simulin( ); maxlinlr( ); learnwh( ); trainwh( ); adaptwh( ); newlind( ); newlin( ); dotprod( ); netprod( ); normprod( ); initwb( ); trainwb( ); adaptwb( );initlay( ); initzero( ); sse( )

3）BP 神经网络函数

tansig( ); logsig( ); deltatan( ); deltalin( ); deltalog( ); learnbp( ); learnbpm( ); learnlm( ); initff( ); trainbp( );trainbpx( ); trainlm( ); simuff( ); newff( ); newfftd( ); newcf( ); nwlog( ); sumsqr( ); errsurf( ); plotes( ); plotep( );ploterr( ); barerr( ); purelin( ); sse( )

4）径向基神经网络函数

dist( ); radbas( ); solverb( ); solverbe( ); simurb( ); newrb( ); newrbe( ); newgrnn( ); newpnn( ); mse( ); ind2vec( ); vec2ind ( )

5）竞争学习神经网络函数

compet( ); nngenc( ); nbdist( ); nbgrid( ); nbman( ); plotsm( ); initc( ); trainc( );simuc( ); newc( ); dist( ); nitsm( ); learnk ( ); learnis ( ); learnos ( ); learnh ( ); learnhd( ); learnsom( ); plotsom( ); trainsm( ); simusm( );newsom( ); mandist( ); linkdist( ); midpoit ( ); negdist( )

6）LVQ 神经网络函数

initlvq( ); trainlvq( ); simulvq( ); newlvq( ); learnlvq( ); plotvec( )

7）Elman 神经网络函数

initelm( ); trainelm( ); simuelm( ); newelm( )

8）Hopfield 神经网络函数

satlin( ); satlins( ); newhop( ); solvehop( ); simuhop( )

9）生成神经网络的模块化描述函数

gensim( )

10）启动神经网络编辑器命令

nntool

2. 模糊逻辑控制工具箱函数

1）模糊推理系统的建立、修改与存储管理函数

newfis( ); readfis( ); getfis( ); writefis( ); showfis( ); setfis( ); plotfis( )

2）模糊语言变量及其语言值函数

addmf( ); rmvar( )

3）模糊语言变量的隶属度函数

plotmf( ); addvar( ); rmmf( ); gaussmf( ); gauss2mf( ); gbellmf( ); pimf( ); sigmf( ); psigmf( ); dsigmf( ); trapmf( ); trimf( ); zmf( ); mf2mf( )

4）模糊规则的建立与修改函数

addrule( ); parsrule( ); showrule( )

5）模糊推理计算与去模糊化函数

evalfis( ); defuzz( ); gensurf( )

6）模糊神经函数

anfis( ); genfis1( ); anfisedit

7）模糊聚类函数

fcm( ); subclust( ); genfis2( )

8）启动模糊推理系统编辑器命令

fuzzy

3. 模型预测控制工具箱函数

1）系统模型辨识函数

autosc( ); scal( ); rescal( ); wrtreg( ); mlr( ); plsr( ); imp2step( ); validmod( )

2）模型建立与转换函数

ss2mod( ); mod2ss( ); poly2tfd( ); tfd2mod( ); mod2step( ); tfd2step( ); ss2step( ); mod2mod( ); th2mod( );addmd( ); addmod( ); addumd( ); paramod( ); sermod( ); appmod( )

3）动态矩阵控制设计与仿真函数

cmpc( ); mpccon( ); mpccl( ); mpcsim( ); nlcmpc( ); nlmpcsim( )

4）基于 MPC 状态空间模型的预测控制器设计函数

scmpc( ); smpccl( ); smpccon( ); smpcest( ); smpcsim( )

5）系统分析与绘图函数

mod2frsp( ); smpcgain( ); smpcpole( ); svdfrsp( ); mpcinfo( ); plotall( ); plotfrsp( ); ploteach( ); plotstep( )

6）模型预测工具箱通用功能函数

abcdchkm( ); cp2dp( ); c2dmp( ); dantzgmp( ); dareiter( ); dimpulsm( ); dlsimm( ); d2cmp( ); mpcaugss( ); dlqe2( ); mpcparal( ); mpcstair( ); parpart( ); ss2tf2( ); tf2ssm( )

参考文献

1 蔡自兴编著. 智能控制. 北京：电子工业出版社，2004

2 李国勇, 谢克明编著. 控制系统数字仿真与CAD. 北京：电子工业出版社，2003

3 张仰森编著. 人工智能原理与应用. 北京：高等教育出版社，2004

4 李国勇. 输入受限的隐式广义预测控制算法的仿真研究. 系统仿真学报，2004，Vol.16，No.7

5 徐丽娜编著. 神经网络控制. 北京：电子工业出版社，2003

6 李国勇. 一种新型的模糊PID控制器. 系统仿真学报，2003，Vol.15，No.10

7 王洪元, 史国栋主编. 人工神经网络技术及其应用. 北京：中国石化出版社，2003

8 李国勇.基于预测控制的电厂取水口物模实验的仿真研究. 系统仿真学报，2002，Vol.14，No.4

9 飞思科技产品研发中心编著. MATLAB6.5 辅助神经网络分析与设计. 北京：电子工业出版社，2003

10 李国勇, 谢克明. 隐式广义预测自校正控制算法的混合控制研究. 系统工程与电子技术，1998，Vol.20，No.6

11 王士同主编. 人工智能教程. 北京：电子工业出版社，2001

12 李国勇. 受约束动态矩阵控制算法的仿真研究. 太原理工大学学报，2004，Vol.35，No.1

13 光建武编著. 神经网络技术及应用. 北京：中国铁道出版社，2000

14 李国勇. 输入受限的广义预测控制算法的鲁棒性. 太原理工大学学报，2004，Vol.35，No.6

15 杨丽娟, 李国勇. 基于高斯函数的非线性自适应 PID 控制器设计. 华北工学院学报，2003，Vol.24

16 李国勇. 一种改进的自适应PID控制器. 太原理工大学学报，2003，Vol.34，No.1

17 王顺显, 舒迪前编著. 智能控制系统及其应用. 北京：机械工业出版社，1999

18 李国勇. 输入受限的广义预测控制算法的稳定性. 华北工学院学报，2004，第4期

19 李士勇编著. 模糊控制神经控制和智能控制论. 哈尔滨：哈尔滨工业大学出版社，1998

20 李国勇. 系统仿真及机辅分析与设计课程的教改探讨. 太原理工大学学报（社科版），2002，Vol.20

21 楼顺天等编著. 基于 MATLAB 的系统分析与设计——神经网络. 西北电子科技大学出

版社，1998

22 李国勇. 一种复合 PID 控制器. 华北工学院学报，2003，Vol.24

23 施阳, 李俊编著. MATLAB 语言工具箱——TOOLBOX 实用指南. 西安：西北工业大学出版社，1998

24 李国勇. 广义预测控制的稳定性与鲁棒性. 自动化学会论文集，兵器工业出版社，1997

25 孙增圻. 智能控制理论与技术. 清华大学出版社，1997

26 李国勇. 橡胶硫化单片机控制仪. 工业仪表与自动化装置，1997

27 余雪丽等编著. 神经网络与实例学习. 北京：中国铁道出版社，1996

28 李国勇. GPC 与 DMC 混合控制算法. 自动化学会论文集. 兵器工业出版社，1997

29 舒迪前编著. 预测控制系统及其应用. 北京：机械工业出版社，1996

30 李国勇, 谢克明. 一种多变量广义预测自校正解耦控制算法. 中国控制会议论文集.中国科学技术出版社，1995

31 谢克明, 李国勇. 中储式制粉系统解耦控制系统. 太原工业大学学报，1994，Vol.25，No.2

32 李国勇，谢克明. 广义预测多变量隐式自校正控制算法的研究. 中国控制会议论文集. 中国科学技术出版社，1994

33 熊淑燕, 李国勇. 一种改进的隐式广义预测自校正控制算法. 太原工业大学学报，1992，Vol.23，No.2

34 席裕庚. 预测控制. 北京：国防工业出版社，1993

35 李国勇, 谢克明. 一种实用的粮仓温度检测系统. 工业仪表与自动化装置，1996

36 杨丽娟, 李国勇. PLC 在磁场压力机中的应用. 太原理工大学学报，2004，Vol.35，No.1

37 阎平凡, 张长水编著. 人工神经网络与模拟进化计算. 北京：清华大学出版社，2000

38 杨丽娟, 李国勇. 微机磅房测量管理系统. 自动化应用技术，2001

39 Rouhani R, Mehra R K. Model Algorithmic Control (MAC). Basic Theoretical Properties. Automatica,1982,18(4):p401-414

40 Clark,D.W.el al,Generalized Predictive Control-Part1.The Basic Algorithm; Part2.Extensions and Interpretations,Automatia,Vol.23 No.2,1987

41 Garcia.C.E.et al, Model Predictive Control:Theory and Practice a Survey, Automatica,Vol.25, No.3,pp.335-348,1989

42 The MathWorks,Inc.http://www.mathworks.com,2005